Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Mesterséges Intelligencia Tanszék

Péter Brezovcsik

FACE ATTRIBUTE RECOGNITION USING EFFICIENT NEURAL NETWORKS

KONZULENS

Dr. Bálint Gyires-Tóth

BUDAPEST, 2025

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Brezovcsik Péter**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2025. 03. 18.

...…….

Brezovcsik Péter

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Recent advancements in deep learning have enabled the development of high-quality computer vision models capable of accurately classifying, segmenting, and detecting objects in images. Face attribute recognition, a specific image classification task, has numerous real-world applications, ranging from facial recognition and emotion classification to personal identification. Many of these use cases are implemented on edge devices, such as mobile phones or IoT devices, which are limited in computational resources. Therefore, the use of efficient neural networks is critical. Over the years, various efficient, lightweight neural architectures, such as MobileNets and EfficientNet, have been developed to operate effectively on resource-constrained devices.

The goal of the thesis is to develop models based on existing efficient neural network architectures to classify different facial attributes from face images.

1 Introduction

Image classification tasks can be divided into several categories, like binary classification, where the task is to classify the image into one of two categories, multi-class classification where the image is assigned to one of several categories or multi-label classification, in which case multiple labels are assigned to an image.

Facial attribute recognition is a multi-label image classification task and has gained significant attention because of its widespread applications. The goal of facial attribute recognition is to predict multiple attributes in a given facial image, like hair color, gender and other facial characteristics. This can be challenging due to the wide variety of facial appearances. These extracted features can be used in various applications, including image retrieval, facial verification and identification.

Throughout the years, demand has grown for efficient neural networks that can work on resource-constrained devices like mobile phones and IoT devices like cameras. These lightweight networks made it possible to create facial attribute recognition and person identification models that can infer in these resource-limited environments.

The aim of this work is to experiment with various efficient neural network architectures and compare their performance in the domain of facial attribute recognition.

2 Background

In this section, I review the basic building blocks of deep-learning models, with a big emphasis on convolutional neural networks.

2.1 Basics of Deep Learning

2.1.1 MLP

One of the most important building blocks of modern neural networks is the multi-layer perceptron. It consists of several fully connected layers that are made up of nodes called neurons, these neurons calculate the weighted sum of its input. The purpose of the MLP is to approximate a function. It receives an input and maps it into an output by using weights or parameters that result in the best function approximation. In a supervised learning scenario, when the network makes a prediction during training, the output and the expected output are compared, this is called the loss function. The goal of the training is to minimize the loss by adjusting the parameters of the network. Gradient descent-based methods are the most popular ones to optimize the network, such methods are Adam or SGD. The gradient of the loss function is calculated with respect to the parameters and then this error is propagated backward layer by layer and the parameters are adjusted to reduce loss. When these error gradients become too big (exploding gradients) or too small (vanishing gradients) the network becomes unstable. To make the network more stable and faster, normalization is used. These methods map the error gradients into a defined range. Such methods are Batch Normalization and Layer normalization. Several other components are used in a neural network, like activation functions. When a neuron calculates the weighted sum of the inputs, its output goes through a non-linear activation function, which decides if the neuron should activated or not [22][28].

The simplest MLPs are the feed-forward neural networks. In these networks, the information flows forward, from input to output, without any loop or recurrent connections. When a feed-forward network contains recurrent connections, it is called a recurrent neural network.

2.1.2 Multi-label Classification

In a multi-label image classification problem, the model should decide, whether a label is present on the image or not. When a model makes its prediction, it outputs an array of probabilities. The role of the threshold is to specify a value at which, if the probability of a given label is bigger, then the label is present in the picture, if it is smaller then it is not present in the image.

There are three types of thresholds. Global threshold, label-wise threshold and instance-wise threshold [9]. A threshold is global when it's a single value and all the labels will be assigned to the image based on this value. A label-wise threshold is when there are different thresholds for each label. A threshold is instance-wise when there are separate thresholds for each sample.

Global, label-wise and instance-wise thresholds can be divided into two further subcategories. Whether a threshold is global, label-wise or instance-wise, it can be fixed or adaptive. A fixed threshold means that the value of the threshold doesn't change in the learning phase of the model. A threshold is adaptive when the value of the threshold changes over time based on the quality of the model's predictions.

2.2 Convolutional Neural Networks

For computer vision tasks, one of the most used neural network architecture is the convolutional neural network. It expects images or three-dimensional inputs and uses Convolution Layers to extract features and then these features are fed into fully-connected layers for classification. In this chapter, I describe the layers and concepts used in a Convolutional Neural Network.

2.2.1 Convolutional Layer

The core element of a CNN is the Convolutional Layer. It extracts features from the input volume, such as shapes and edges and when there are several layers these extracted features become more complex. This layer consists of several learnable filters or kernels, each having a spatial dimension and a depth (mostly the same as the depth of the input). Every kernel will traverse the input volume with a specific stride value and perform an elementwise multiplication between the kernel weights and the corresponding input pixels, then sum it together and output a two-dimensional feature map.

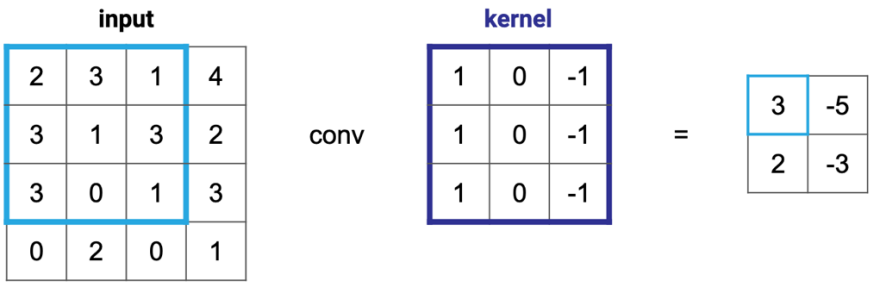

Figure 1: Convolution operation [29]

The number of filters determines the depth or the number of channels/feature maps of the output. The size of the filter controls the spatial dimensions of the output. The stride value indicates how many pixels the kernel shifts at each step. Increasing the stride value is a way of reducing the spatial dimension. The most used filter sizes in practice are 3x3 and 1x1 filters. The latter is also called pointwise convolution, it is used to increase or decrease the number of output channels while preserving the input's spatial dimension. The reason why 3x3 filters are more popular than 5x5 or 7x7 filters is that if two 3x3 filters are stacked the effective receptive field is 5x5 and if three 3x3 filters are stacked the effective receptive field is 7x7, while having fewer parameters [1].

2.2.2 Receptive Field

The receptive field is the region of the input image that the network depends on when creating an output feature [24]. Pixels outside the receptive field do not affect the final feature. Therefore, the receptive field should be controlled to cover all of the relevant area in the input image, this is also called the theoretical receptive field. Because not all pixels in the receptive field contribute equally to the output feature [2], the area of the theoretical receptive field that affects the output feature is called the effective receptive field [2].

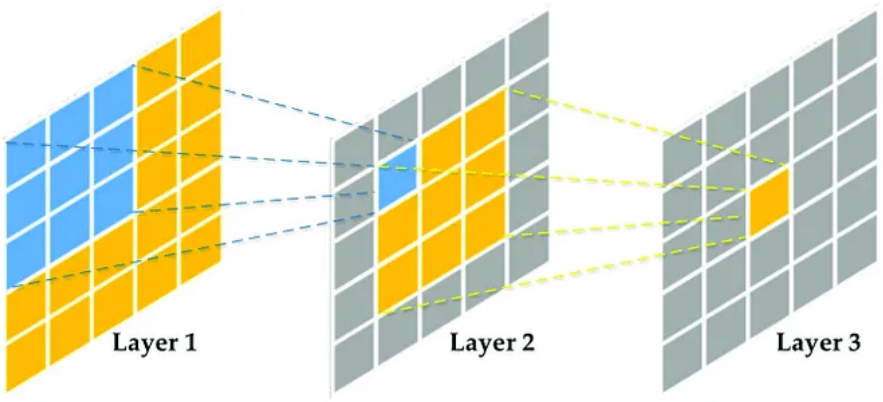

Figure 2: Receptive field [30]

To increase the receptive field, there are several options, one is to increase the number of convolutional layers. Another method is to use pooling layers, dilated convolutions or increase the stride.

2.2.3 Pooling Layer

Pooling is a sub-sampling method whose goal is to reduce the input's spatial dimension but at the same time preserve the relevant information. Like the Convolutional Layer, it also has a filter and a stride, but it does not have any learnable parameters. There are several types of pooling layers, but the most common are the maximum pooling and average pooling. Maximum pooling selects the biggest values from the region that the filter covers. Average pooling averages the values from the area that the filter covers. There is also a pooling technique called global average pooling. This was proposed by Lin. et al. [8], and is used in many convolutional networks to replace or reduce the number of fully-connected layers at the end of the network. This is, in essence, an average pooling layer with a filter size equal to the input's spatial dimension and it produces a scalar value from each feature map. This approach is more robust to spatial translations [8] and overfitting is avoided because there are no learnable weights in this layer.

2.2.4 Group Convolution

To produce a feature map, a filter has to have the same depth as the input volume, this is not an efficient approach when there are several output feature maps. Group convolutions split the input volume into several groups by the input channels and for each group, there is a set of filters that produces a feature map. This approach uses fewer

parameters but each output feature map only relates to the input channels within the group [5]. Group convolutions were originally proposed by Krizhevsky et al. [Alexnet] to split the convolutions between two GPUs. When each input channel forms a group it is called depthwise convolution which is an efficient way of using convolution.

2.2.5 Depthwise-separable Convolution

Depthwise-separable Convolution [11] is a factorization of the classical convolution into two layers, a depthwise convolution and a pointwise convolution. As mentioned in Chapter 2.2.4, a depthwise convolution is when there is a single filter for every input channel of the volume. In this case, the output feature map only relates to one input kernel group. To achieve the same expressive power as a normal convolution, the depthwise convolution is followed by a pointwise convolution which combines the outputs of the depthwise convolution. In this way of splitting the convolution into two parts, we can reduce the model size and amount of computations needed to create an output feature while preserving the same receptive field.

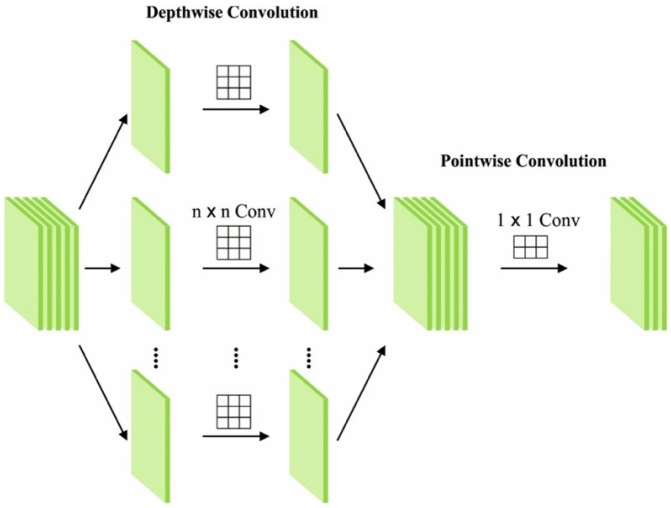

Figure 3: Depthwise-separable convolution [14}

3 Related work

3.1 Efficient Architectures

One disadvantage of convolutional neural networks is that they are computationally expensive and require high-performance hardware like GPUs and TPUs. Several techniques and small networks were proposed to speed up CNNs performance, both the backpropagation and the feedforward pass.

There are several approaches to obtaining a small network, one is to reduce the number of parameters that the model has. Often, the parameters in CNNs have high redundancy and have the filters represent similar patterns [15]. This increases the required computation and also decreases the learning capacity. For example, to reduce the redundancy of the filters, Jin et al. [15] proposed Flattened CNNs that separate the 3D convolution into three consecutive 1D filters. Another method to train small models is knowledge distillation [19]. A large model is used to train a smaller network, so the knowledge of the large model is transferred to the smaller one, which is more suitable for deployment.

3.1.1 MobileNet

MobileNets are a class of efficient model architectures for mobile and embedded architectures proposed by Howard et al. [11]. It uses 3x3 depthwise separable convolutions to reduce the size of the model and increase speed by using 8 to 9 times less computations than standard convolutions. Every layer, except the last one, is followed by a batchnorm and a ReLU. To reduce the spatial dimension of the image, MobileNet uses strided convolutions in the depthwise layers, and then a final average pooling reduces the spatial dimension into 1x1. From all the layers, the 1x1 convolutions have the most parameters and account for 95% of the computation time [11]. To further increase the performance of the MobileNets, two global hyperparameters adjust the trade-off between latency and accuracy. One is called the width multiplier, which is a constant that ranges between 0 and 1, with 1 being the baseline MobileNet. Its task is to thin the network by adjusting the number of input and output channels uniformly. It quadratically reduces the number of parameters and computational cost. The other hyperparameter is called the resolution multiplier. It is also a constant between 0 and 1 and reduces the cost of

computation quadratically, but in contrast to the width multiplier, its value is set implicitly by setting the input resolution.

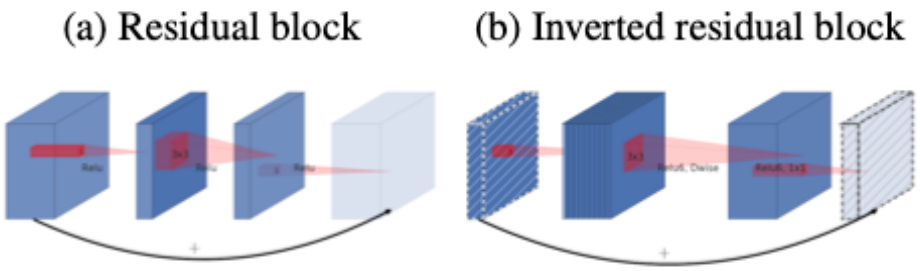

Figure 4: Residual and Inverted Residual blocks [10]

MobileNetV2 proposed by Sandler et al. [10], is an improved version of the MobileNet. It uses a module called an inverted residual block, which is an inversion of the standard residual block. It first increases the number of channels in the input using a 1x1 convolution then applies a 3x3 depthwise convolution and then with another 1x1 convolution, it reduces the channel dimension. At last, the input and output are added together. It reduces memory usage during inference and makes it suitable for mobile applications.

3.1.2 ShuffleNet

ShuffleNet is a highly efficient network architecture proposed by Zhang et al. [5], designed for devices with very limited computational resources. Because pointwise convolutions are expensive, they limit the number of channels in a small network. ShuffleNet uses a pointwise group convolution and the channel shuffle operation to overcome this problem. The usage of the pointwise group convolutions significantly reduces the cost of computation. The channel shuffle operator enables inter-group information flow and also improves accuracy. Its main building block is the ShuffleNet Unit, a residual block that contains channel shuffle and convolution operators.

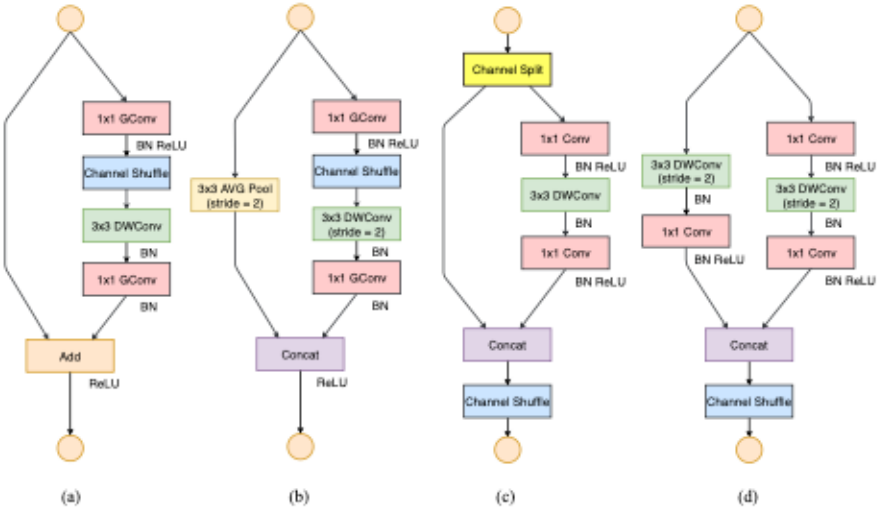

Figure 5: [6]

An improved version of ShuffleNet, proposed by Ma et al. [6], introduces a new operation called channel split, which splits the number of input channels into two parts at the beginning of the ShuffleNet Unit. One branch of the split remains as an identity of the original input, the other goes through convolutions. It also omits pointwise group convolutions for two reasons. One is that channel split already creates two groups, and the other is that extensive group convolutions require fewer FLOPs than dense convolutions but have high memory access cost (MAC) because of the increased number of channels. Because of this high MAC/FLOPs ratio, it also uses fewer element-wise operations like ReLU or AddTensor.

3.1.3 EfficientNet

EfficientNet is an efficient network architecture and a dimension scaling technique [18]. The authors proposed a compound scaling method that uniformly scales the depth, width and resolution of the network. The method uses a user-specified coefficient that controls the remaining resources for scaling and three constants that are determined by a grid search on the original model. These three constants specify how to assign the resources to the different dimensions.

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

Equation 1: Compound scaling method [18]

This balanced scaling of the network depth, width and resolution leads to better accuracy and efficiency. The authors used neural architecture search to develop a new model architecture to maximize the effectiveness of the model scaling method. The network is composed of inverted residual blocks, which are optimized with squeeze-and-excitation [3]. The base model, EfficientNet-B0, reached 77.1% Top-1 accuracy on the ImageNet dataset with only 5.4 million parameters. For comparison, ResNet-50 with 26 million parameters reached 76.0% Top-1 accuracy on the same dataset.

An upgraded version of the EfficientNet was proposed by Tan and Le [17], named EfficientNetV2. It reduces the training speed and improves parameter efficiency. They used a scaling and neural network search to develop this model family and they also introduced the fused inverted residual block that replaces the pointwise and depthwise convolution to a single 3x3 for scaling up the channel dimension.

3.2 Face Attribute Recognition

Throughout the years, many different methods and architectures have been proposed for facial attribute recognition (FAR) tasks. In this section, I review a few proposed methods.

For example, Sharma and Foroosh [4] proposed a CNN micro-architecture for FAR called Slim-CNN. They incorporated several structural elements into the model, like skip-connection and multiple branches, while achieving comparable results to those of state-of-the-art models.

Most FAR models are trained in a supervised learning fashion. Therefore, they require labeled datasets. These datasets can be laboratory-controlled, where the images are taken in a controlled environment or can be „in the wild" datasets, which are datasets that contain images taken in natural environments. There are many popular datasets for facial attribute recognition, for example, LFW [13] (Labeled Faces in the Wild) contains more than 13,000 images of faces, the UTKFace dataset [26] is a large-scale dataset with over 20,000 face images with annotations of age, gender and ethnicity. FairFace dataset [16] contains more than 100,000 images from seven different race groups, specifically designed for racially fair race group classification. CelebA [21] dataset is another large-scale dataset that contains hundreds of thousands of images labeled with 40 attributes.

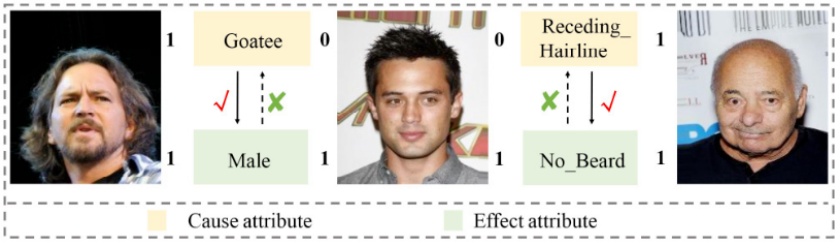

Figure 6: Causal relationship of attributes [7]

FAR models require a large dataset, like the CelebA dataset, but for many applications, the available amount of data is limited. To overcome this problem, [12] used a self-supervised method to extract spatial and semantic information from unlabeled data and then finetuned the model on a limited dataset. To improve the performance of a FAR model and a multi-label classification task in general, many methods exploit the correlation between the labels. Many methods use group learning, which is a technique to group attributes with strong correlations. For example, Emily et al. [25] created nine groups of labels based on the location of the attributes. Mao et al. [20] split the facial

attributes into two sets, objective and subjective. They also use the facial landmark locations to improve prediction. Instead of correlation, [7] uses the causality of the attributes and classifies them into causal and effect attributes. They also use a wavelet-scattering transformation (WST), which can be seen as a CNN with predetermined, nonlearnable kernels and design a WST block that uses the Squeeze-and-Excitation unit. WST and wavelet-based methods efficiently extract image features in the frequency domain.

4 Own Work

In this chapter, I provide the results of the experiments conducted on different network architectures.

4.1 Basic settings

For the backbone of the models, I used the three previously mentioned in Section 2, namely ShuffleNetV2, MobileNetV2 and EfficientNet. All three models were pretrained on the ImageNet-1k dataset, which contains millions of training images ranging over 1000 classes. As a deep learning framework, I used PyTorch.

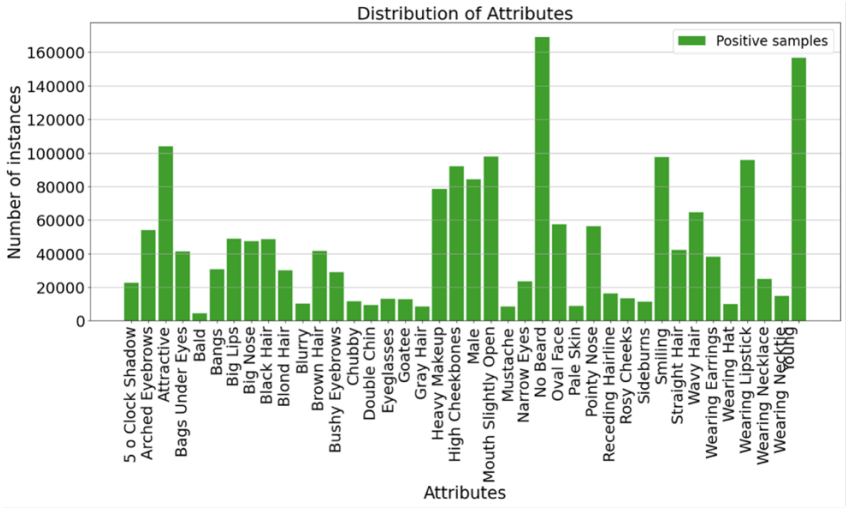

Figure 7: Distribution of positive labels

For a dataset, I chose the CelebA dataset. CelebA dataset is a large-scale dataset with more than 200,000 images of faces and 40 attributes, such as hair color, beard, moustache, etc. It also includes bounding box and facial landmark annotations. The images were resized, center-cropped to 224x224 pixels, then the pixel values were scaled to be between 0 and 1, and finally, the images were normalized. One of the major problems of this dataset is that it's highly imbalanced. A dataset is imbalanced when the distribution of the samples and their corresponding labels are uneven over the data space [27]. There are three types of data imbalance in a multi-label classification problem: imbalance within labels, imbalance between labels, and imbalance among label sets [27]. The CelebA dataset is imbalanced between labels, which means that the number of positive samples for a given label is much higher than the number of positive samples for

another label. It is also imbalanced within its labels because many labels contain an extremely high number of negative examples and a low number of negative samples and vice-versa. The distribution of labels of the CelebA dataset is shown in Figure 6. Because of computational limitations, I only used a subset of the original CelebA dataset, containing around 5000 images to fine-tune the different models.

For training, I used the Adam optimizer with a learning rate of 0.0001 and a batch size of 64. The models were trained over 20 epochs with binary cross entropy as the loss function and with a classification threshold of 0.5.

4.2 Evaluation metrics

In a multi-label classification task when evaluating the model's output each label can be evaluated as True Positive, True Negative, False Positive or False negative.

A prediction is True Positive when the model correctly predicts that the given label is present on the image. True Negative prediction means that the model predicts correctly that a label is not on the image. A prediction is a False Positive when the model incorrectly assigns a label to the image and a False Negative when it incorrectly predicts that the given label is absent from the image. From these four values, further evaluation metrics can be calculated like recall and precision scores. In a multi-label classification problem, all of these scores are calculated for each label and then averaged.

I used five evaluation metrics. The first was accuracy, which in a multi-label classification problem is not trivial. I used subset accuracy, which is the ratio of the correctly predicted labels to all of the labels. However, by itself, it is not as expressive because, for example, in the CelebA dataset, most of the samples only have a handful of positive labels, which means a high accuracy score can be achieved simply by assigning negative labels to most of the attributes. Therefore, I used other metrics like precision score, recall score and f1 score. All of these scores were calculated for each label and then averaged and weighted by the number of true instances for each label. This way of calculating these scores accounts for the label imbalance in the dataset. The last metric I used was label-wise accuracy. This is somewhat similar to the subset accuracy, but it is calculated for each label independently in a minibatch.

4.3 Experiments

The models were trained according to the parameters described in Section 4.1. The evaluation dataset contained 25.000 samples, which weren't used during training. The models were compared in two domains. The first one contains the metrics described in Section 4.2, like precision, recall scores and label-wise accuracy. The second one is the inference time of the models.

4.3.1 CNN with a single classifier head

At first, I used the three models as feature extractors and then a single classification head was attached to the base model. This means that a model predicts a vector of size 40. The classification model contains a dropout layer, with a probability of 0.2, and a fully-connected layer.

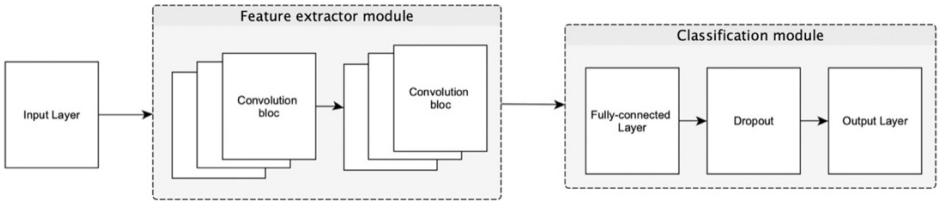

Figure 8: A basic Deep CNN network [23]

The results of the evaluations are shown in Figure 8 and Figure 9. The best results were achieved in every metric with the EfficientNetB0 model.

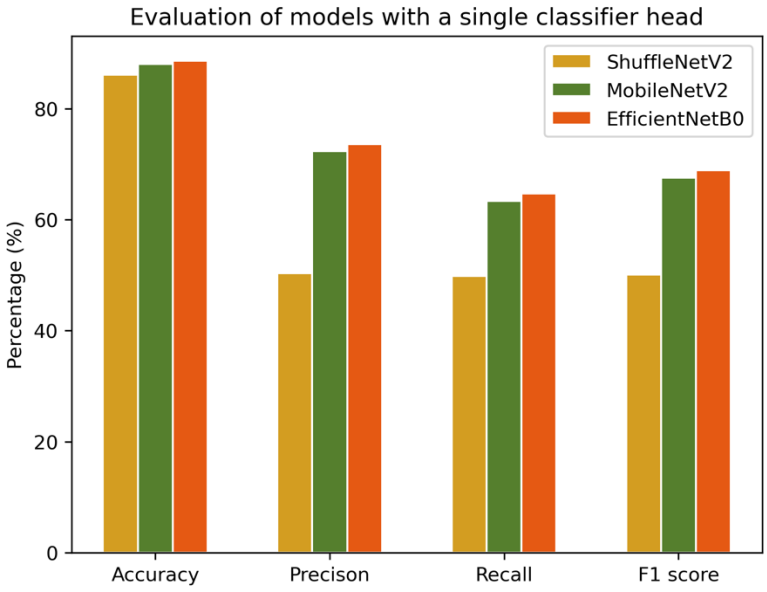

Figure 9: Evaluation metrics of the different models

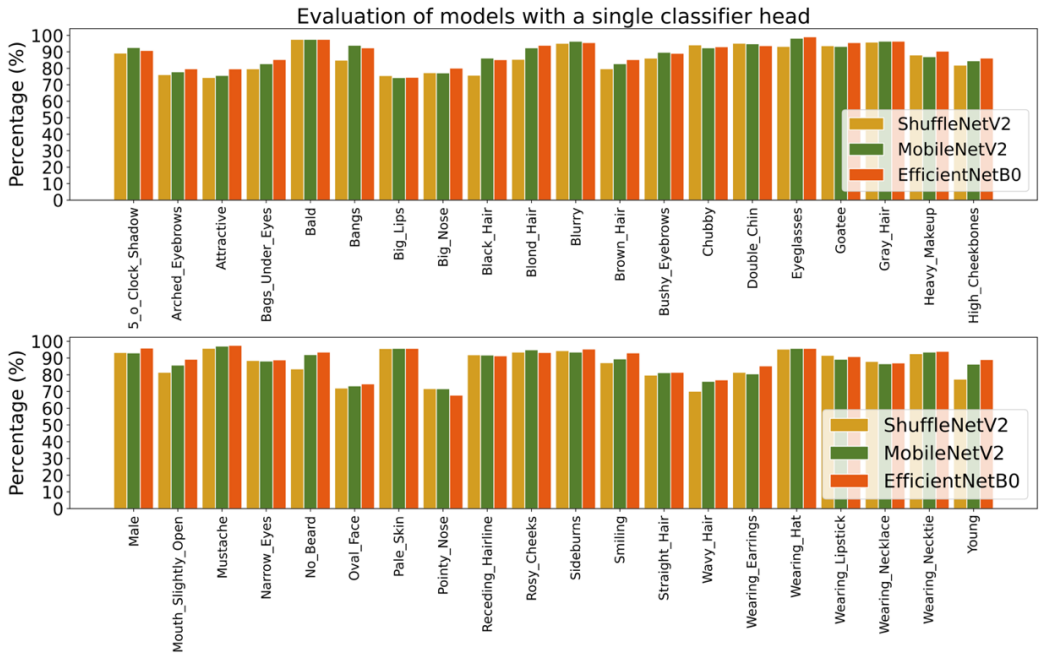

Figure 10: Label-wise accuracies of the models

Higher label-wise accuracies were achieved for attributes with a low number of positive samples and lower for those with a high number of positive samples. Figure 10 shows the relationship between the number of positive labels and the accuracy. This is caused by the imbalance within the labels of the CelebA dataset and therefore, without using any techniques to handle the imbalance nature of the dataset, the accuracy for a given label decreases when the number of positive samples increases for that label.

Commonly used techniques to handle imbalanced datasets are resampling methods. Resampling methods used during data preprocessing. The two main categories of resampling methods are undersampling and oversampling. Undersampling removes samples from the dataset that are associated with the majority of labels. Oversampling generates samples associated with the minority labels. However, in a multi-label classification problem, it is not trivial how to use them because, for example, it is not guaranteed that using undersampling, the distribution of the labels would be uniform. For example, I only used a subset of the original dataset, but the distribution of the attributes is almost the same as in the original dataset. Similarly, collecting new samples does not guarantee that the number of previously majority labels will not increase further.

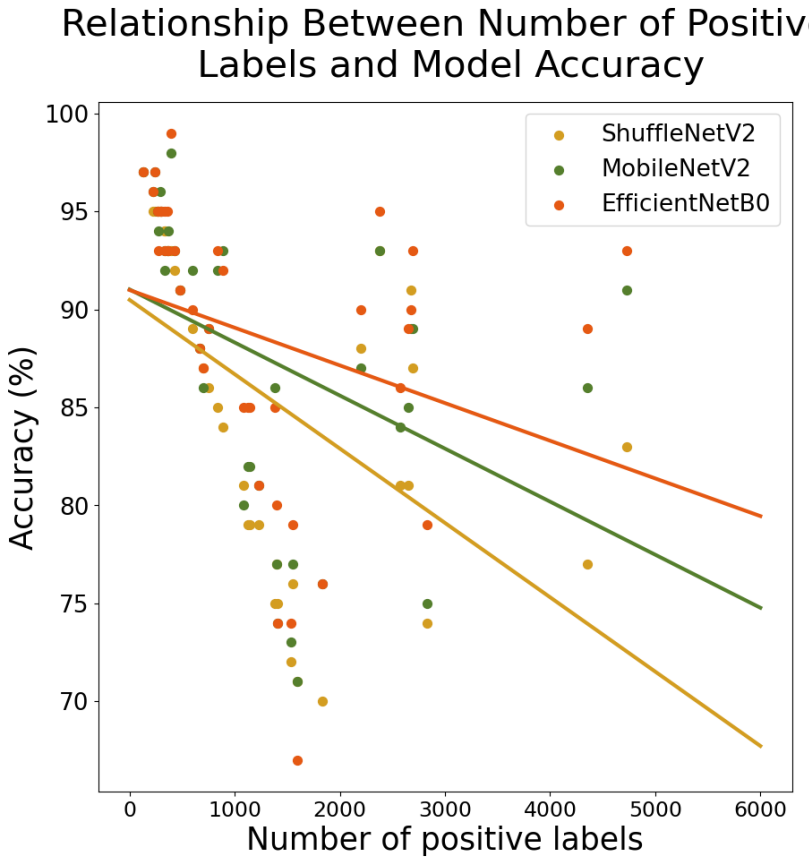

Figure 11: Relationship between the number of positive labels and the model accuracy

Another method to handle an imbalanced multi-label dataset is by adding weight to the positive samples. This weight is the ratio between the negative and positive samples of a label. When calculating the loss function for a label, the binary cross-entropy is multiplied by this weight, this ensures that the calculation of the loss function accounts for the distribution of the labels. When using this method, the relationship between the number of positive labels and the accuracy of the models changes. Specifically, the slope of the regression line is descending at a lower rate.

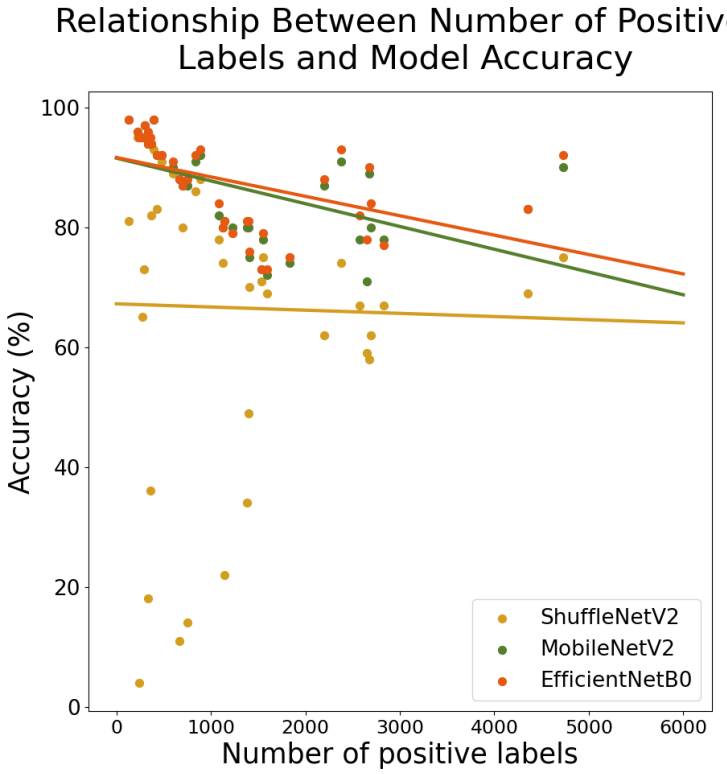

Figure 12: Relationship between the number of positive labels and the model accuracy when the loss is weighted

However, the average label-wise accuracy of the weighted models is 7% lower than the models without loss weighting. This is because even though the EfficientNetB0 and MobileNetV2 achieved similar results to the previous models that did not use any loss weighting, the ShuffleNetV2 achieved lower label-wise accuracy than the ones not using loss weighting. The comparisons of the models are shown in Table 1.

	ShuffleNetV2		**MobileNetV2**		**EfficientNetB0**	
	Not weighted	Weighted	Not weighted	Weighted	Not weighted	Weighted
Accuracy	**86.08**	66.94	**87.99**	87.25	**88.63**	88.01
Precision	50.31	**60.55**	72.36	**72.64**	73.62	**74.58**
Recall	49.83	**51.40**	**63.36**	59.68	**64.72**	62.86
F1 score	50.79	**55.60**	**67.56**	65.52	**68.88**	68.22

Table 1: Evaluation metrics of the models

4.3.2 CNN with multiple classifier heads

The second group of models are using multi-task learning. In this case, 40 subtasks are defined, these are binary classifiers that are arranged in parallel. Their structure is the same as a single output classifier, a dropout layer and a fully-connected layer. I used a hard parameter-sharing approach to MTL, which means that the subtasks are sharing the base model as a feature extractor. I also used the previously mentioned loss function weighting method. A challenge of an MTL network is how to balance or account for the losses of the different tasks. Since in this case, all subtask has the same binary classification task, utilizing the same loss function and also wasn't any attribute grouping, I chose to use two approaches.

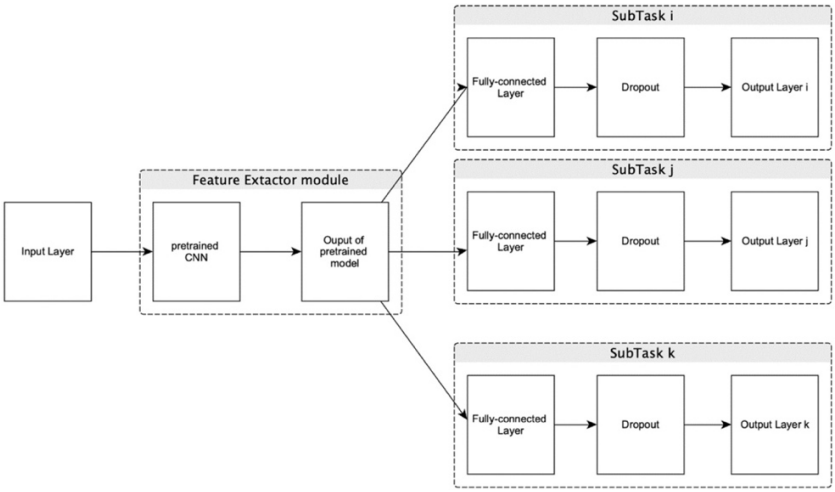

Figure 13: Structure of a multi-output CNN [23]

The first approach only contains one loss function. All 40 classifiers make a prediction, and then the outputs are concatenated. The loss function then uses this 40-element vector to calculate the loss. This calculation of the loss is similar in a way to the single output classifier's because it only calculates the loss for the concatenated vector, but instead of one classifier head making a prediction, 40 different classifier heads make a binary prediction. The evaluation result shown in Figure

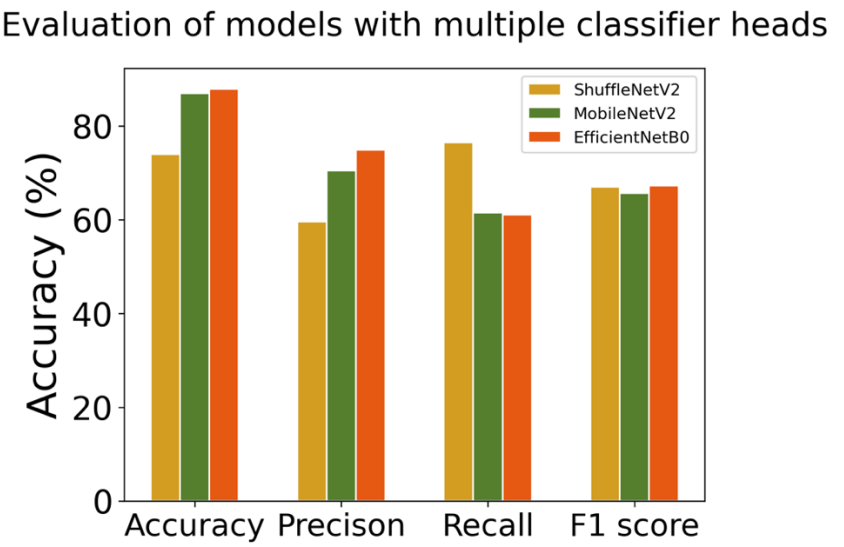

Figure 14: Evaluation results of models with multiple classifier heads

In the second approach, unlike the first, the loss is calculated for each subtask. Then, these losses are summed and averaged. This way of calculating the joint loss, just like the previous one, does not account for the correlation between the labels.

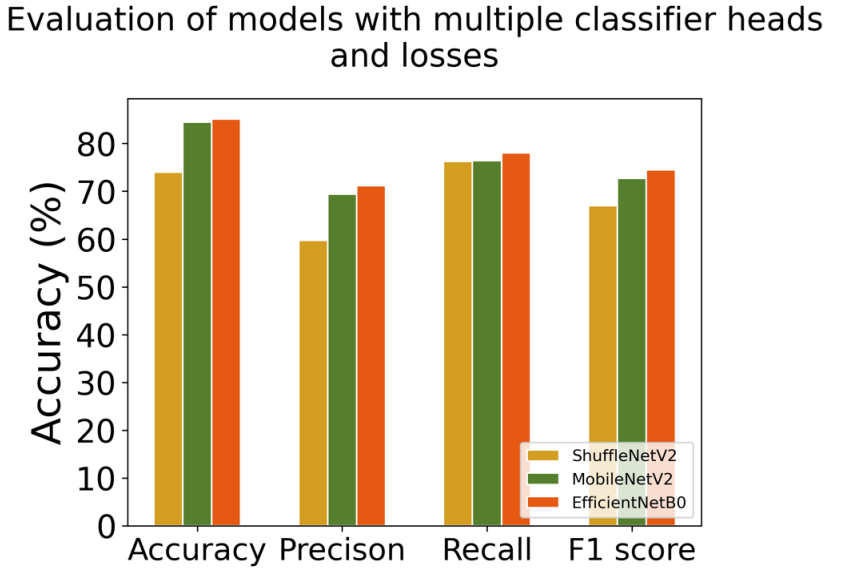

Figure 15: Evaluation results of models with multiple classifier heads and losses

4.3.3 Comparing inference times

In this section, I summarize the inference time of the models. I measured the inference time for a single image that is already preprocessed. Below Table 2 contains the results, for both the single and multiple classifier head models.

	Classifier modul	**Inference time (ms)**
ShuffleNet	Single	53.21
	Multiple	53.87
MobileNetV2	Single	140.50
	Multiple	143.56
EfficientNetB0	Single	185.70
	Multiple	187.94

Irodalomjegyzék

[1] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Apr. 10, 2015, *arXiv*: arXiv:1409.1556. Accessed: Nov. 05, 2024. [Online]. Available: http://arxiv.org/abs/1409.1556

[2] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the Effective Receptive Field in Deep Convolutional Neural Networks,” Jan. 25, 2017, *arXiv*: arXiv:1701.04128. Accessed: Nov. 05, 2024. [Online]. Available: http://arxiv.org/abs/1701.04128

[3] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-Excitation Networks,” May 16, 2019, *arXiv*: arXiv:1709.01507. Accessed: Nov. 19, 2024. [Online]. Available: http://arxiv.org/abs/1709.01507

[4] A. Sharma and H. Foroosh, “Slim-CNN: A Light-Weight CNN for Face Attribute Prediction,” Jul. 03, 2019, *arXiv*: arXiv:1907.02157. Accessed: Nov. 20, 2024. [Online]. Available: http://arxiv.org/abs/1907.02157

[5] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” Dec. 07, 2017, *arXiv*: arXiv:1707.01083. Accessed: Oct. 24, 2024. [Online]. Available: http://arxiv.org/abs/1707.01083

[6] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design,” Jul. 30, 2018, *arXiv*: arXiv:1807.11164. Accessed: Oct. 24, 2024. [Online]. Available: http://arxiv.org/abs/1807.11164

[7] N. Liu, F. Zhang, L. Chang, and F. Duan, “Scattering-based hybrid network for facial attribute classification,” *Front. Comput. Sci.*, vol. 18, no. 3, p. 183313, Jun. 2024, doi: 10.1007/s11704-023-2570-6.

[8] M. Lin, Q. Chen, and S. Yan, “Network In Network,” Mar. 04, 2014, *arXiv*: arXiv:1312.4400. Accessed: Nov. 05, 2024. [Online]. Available: http://arxiv.org/abs/1312.4400

[9] R. Al-Otaibi, P. Flach, and M. Kull, “Multi-label Classification: A Comparative Study on Threshold Selection Methods,” Sep. 2014.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Mar. 21, 2019, *arXiv*: arXiv:1801.04381. Accessed: Nov. 12, 2024. [Online]. Available: http://arxiv.org/abs/1801.04381

[11] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 17, 2017, *arXiv*: arXiv:1704.04861. Accessed: Nov. 06, 2024. [Online]. Available: http://arxiv.org/abs/1704.04861

[12] Y. Shu, Y. Yan, S. Chen, J.-H. Xue, C. Shen, and H. Wang, "Learning spatial-semantic relationship for facial attribute recognition with limited labeled data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11916–11925.

[13] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments," University of Massachusetts, Amherst, 07–49, Oct. 2007.

[14] D.-N. Le, V. S. Parvathy, D. Gupta, A. Khanna, J. J. P. C. Rodrigues, and K. Shankar, "IoT enabled depthwise separable convolution neural network with deep support vector machine for COVID-19 diagnosis and classification," *Int. J. Mach. Learn. & Cyber.*, vol. 12, no. 11, pp. 3235–3248, Nov. 2021, doi: 10.1007/s13042-020-01248-7.

[15] J. Jin, A. Dundar, and E. Culurciello, "Flattened Convolutional Neural Networks for Feedforward Acceleration," Nov. 20, 2015, *arXiv*: arXiv:1412.5474. Accessed: Nov. 12, 2024. [Online]. Available: http://arxiv.org/abs/1412.5474

[16] K. Kärkkäinen and J. Joo, "FairFace: Face Attribute Dataset for Balanced Race, Gender, and Age," Aug. 14, 2019, *arXiv*: arXiv:1908.04913. doi: 10.48550/arXiv.1908.04913.

[17] M. Tan and Q. V. Le, "EfficientNetV2: Smaller Models and Faster Training," Jun. 23, 2021, *arXiv*: arXiv:2104.00298. Accessed: Oct. 27, 2024. [Online]. Available: http://arxiv.org/abs/2104.00298

[18] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Sep. 11, 2020, *arXiv*: arXiv:1905.11946. Accessed: Oct. 16, 2024. [Online]. Available: http://arxiv.org/abs/1905.11946

[19] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” Mar. 09, 2015, *arXiv*: arXiv:1503.02531. Accessed: Nov. 12, 2024. [Online]. Available: http://arxiv.org/abs/1503.02531

[20] L. Mao, Y. Yan, J.-H. Xue, and H. Wang, “Deep Multi-task Multi-label CNN for Effective Facial Attribute Classification,” Feb. 10, 2020, *arXiv*: arXiv:2002.03683. Accessed: Nov. 19, 2024. [Online]. Available: http://arxiv.org/abs/2002.03683

[21] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep Learning Face Attributes in the Wild,” in *Proceedings of International Conference on Computer Vision (ICCV)*, Dec. 2015.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[23] S. Coulibaly, B. Kamsu-Foguem, D. Kamissoko, and D. Traore, “Deep Convolution Neural Network sharing for the multi-label images classification,” *Machine Learning with Applications*, vol. 10, p. 100422, Dec. 2022, doi: 10.1016/j.mlwa.2022.100422.

[24] A. Araujo, W. Norris, and J. Sim, “Computing Receptive Fields of Convolutional Neural Networks,” *Distill*, vol. 4, no. 11, p. 10.23915/distill.00021, Nov. 2019, doi: 10.23915/distill.00021.

[25] E. M. Hand and R. Chellappa, “Attributes for Improved Attributes: A Multi-Task Network for Attribute Classification,” Apr. 25, 2016, *arXiv*: arXiv:1604.07360. doi: 10.48550/arXiv.1604.07360.

[26] Zhang Zhifei, Song, Yang and H. Qi, “Age Progression/Regression by Conditional Adversarial Autoencoder,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017.

[27] A. N. Tarekegn, M. Giacobini, and K. Michalak, “A review of methods for imbalanced multi-label classification,” *Pattern Recognition*, vol. 118, p. 107965, Oct. 2021, doi: 10.1016/j.patcog.2021.107965.

[28] GeeksForGeeks: Multi-Layer Perceptron Learning in Tensorflow https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/ (Accessed: Nov. 27, 2024)

[29] 8th Lights: What Is a Convolution? How To Teach Machines To See Images https://8thlight.com/insights/what-is-a-convolution-how-to-teach-machines-to-see-images
(Accessed: Nov. 8, 2024)

[30] Medium: Receptive Field
https://medium.com/@saba99/receptive-field-1726fe6ea94f
(Accessed: Nov. 8, 2024)