Machine Learning Deployment Assignment 2

Dhruv Verma
Queen Mary University of London
Mile End Rd, Bethnal Green, London E1 4NS
ec211263@qmul.ac.uk / dhruvverma3098@proton.me

1. Introduction

A key aspect of human communication is facial expression, which not only communicates ideas and thoughts but also emotions. There are many fascinating aspects of facial expressions, such as anger, disgust, fear, sadness, sadness, and surprise. This is one of the most fascinating aspects of facial expression. Over the past few years, automatic facial expression recognition/analysis has gained traction due to its wide variety of applications. Because facial expressions are subtle, complex, and variable, the problem remains challenging. The ongoing challenges in the domain of Facial Expression Recognition include the static nature of datasets available which also lack diversity, uncertain environmental factors, on-board processing power for active tracking, etc. This assignment has been designed to investigate the problem domain of Facial Expression Recognition (FER) in real-life environments, i.e., in-the-wild and demonstrate understanding of developing and deploying machine learning approaches for FER.

Applications of FER are versatile and include Access Control, Security and Surveillance, Health and Safety, eKYC, FinTech, etc. A large number of applications indicate a massive data set to validate accurately as this lack of accuracy can lead to critical losses in all previously mentioned fields. Deep Learning is being used as a means of solving problems encountered in this field. We can use the deep neural hierarchy to classify raw data to classify objects that the neural hierarchy has trained or has had experience with.

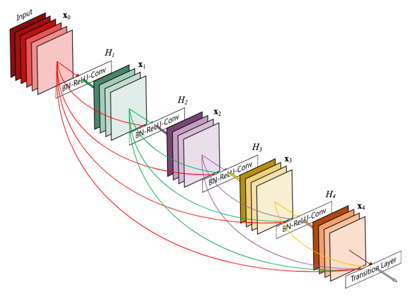
In this assignment, we have used two different neural network architectures to run training, validation and testing on the given dataset. The given dataset is part of Aff-Wild2 [3, 5–10, 12, 13, 16] which is an extension of Aff-Wild dataset. The dataset consists of six main classes Happiness, Sadness, Anger, Disgust, Fear and Surprise. The provided dataset is imbalanced and thus we have balanced the dataset in the preprocessing portion of the assignment. Instead of going for the originally proposed *weight imbalance* techniques, we decided it was better to use PyTorch's pre-trained models [15]. The two models we have decided to use in this assignment keeping in mind the restrictions and hardware limitations on us are DenseNet [2] [17] and VGG11 [1].

The assignment is divided into 4 different sections. **Task 1** is our main machine learning model which gave us the highest testing accuracy, **Task 2** deals with our baseline model with different architecture and hyper-parameters and how it performs in comparison to our main model. **Task 3** is an ablation study performed on our main model and tuning the best model possible. In **Task 4** we perform testing on a separately provided production dataset and show our conclusions.

2. Task 1: Main Machine Learning Model DenseNet121

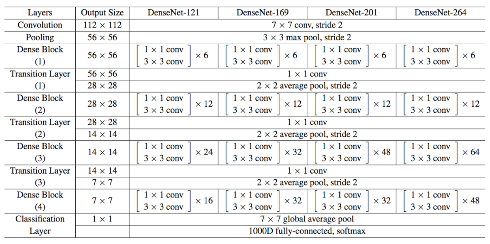
For the main machine learning model, we have chosen DenseNet.DenseNet is composed of Dense blocks, within those blocks, the layers are connected with high density: Each layer gets the input from previous layers' output features. This continuous and massive reuse of residuals creates deep governance because every layer receives more regulation from the previous layer and thus loss function will react accordingly. Thus, the network ends up becoming powerful in terms of accuracy and classification. [2] From the diagram below fig. 1 from [2] we can see the Dense blocks in DenseNet.

Figure 1. Block Diagram of DenseNet



Simply, DenseNet's convolution generates less number of feature maps. DenseNet has a lower need for wide layers, unlike ResNet [17] because as layers are connected with high density, there is little repetition in the learnt features. Layers of dense blocks share a snippet of combined knowledge. The no. of output feature maps of a layer is defined as the growth rate. Eventually, the growth rate controls how much information is refreshed in each layer and contributes to the global overall. Below is the architecture diagram of DenseNet fig. 2 [2].

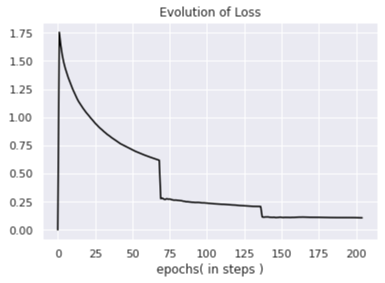
Figure 2. Architecture Diagram of DenseNet



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

In our main machine learning framework, we have used a pre-trained model, DenseNet121 provided by PyTorch [15]. Due to the nature of DenseNet to consume huge amounts of VRAM, we were limited to using the most basic pre-trained model, i.e, DenseNet121. The model was initialised and the classifier modified to output 6 classes, i.e, 'HAPPINESS', 'DISGUST', 'ANGER', 'SADNESS', 'SURPRISE' and 'FEAR'. The hyper-parameters used were **Learning Rate = 2e-5**, **Batch Size = 16** and **Epoch = 3.** The constraints on the hyper-parameters were due to the hardware limitations but the model managed to achieve satisfactory results. The following plots represent the evolution of the loss curve (Fig. 3) and training accuracy and training F1 curve (Fig. 4).

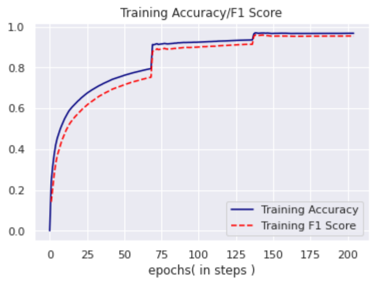
Figure 3. Block Diagram of DenseNet



The data over each epoch can be seen in table 1 and and our testing accuracy and F1 score can be seen in table 2 :

The provided dataset gives satisfying results on DenseNet121 with a testing F1 score of **96.2** and a testing accuracy of **95.99** in just 3 epochs since we are using a pre-trained model. In the final epoch, we can also observe that training accuracy and F1 score have stabilised. The normalised confusion matrix can be seen in Fig 5, the diagonal represents the accuracy score for each class. Thus, we can say we have a model with satisfying accuracy.

Figure 4. Block Diagram of DenseNet



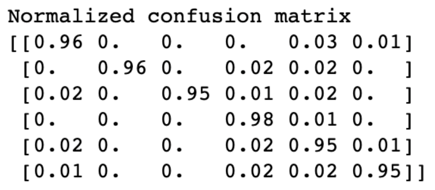
Epoch	Training	Validation	Validation F1
1	0.796307	0.907609	0.909961
2	0.935208	0.943340	0.941593
3	0.967680	0.953862	0.953311

Table 1. Data in each epoch

Testing Accuracy	Testing F1 Score
0.95999	0.96239

Table 2. Main Model Testing Data

Figure 5. Confusion Matrix of DenseNet



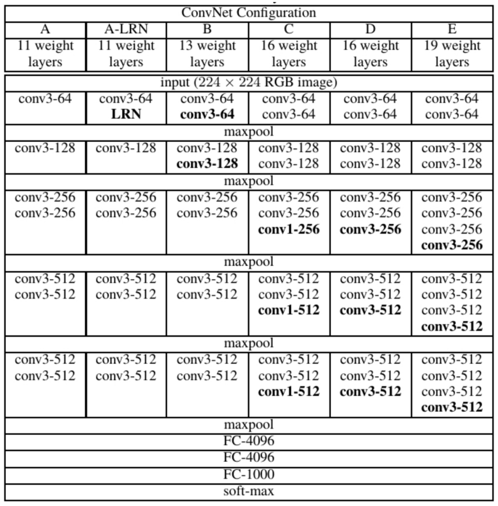
```
Normalized confusion matrix
[[0.96 0.   0.   0.   0.03 0.01]
 [0.   0.96 0.   0.02 0.02 0.  ]
 [0.02 0.   0.95 0.01 0.02 0.  ]
 [0.   0.   0.   0.98 0.01 0.  ]
 [0.02 0.   0.   0.02 0.95 0.01]
 [0.01 0.   0.   0.02 0.02 0.95]]
```

3. Task 2: Baseline Machine Learning Model VGGNet

For the baseline machine learning model, we have chosen VGG11 of the VGGNet architecture. The VGG architecture is an improvement on existing AlexNet [1], as it improves upon the depth of CNNs. VGG11 as the name implies is an architecture of 11 weighted layers, the weights represent the strength of connectivity between units in adjacent layers of a network. Weights near zero indicate that the output won't change if the input is changed. The architecture of VGG11 can be seen in fig. 6.

The model implementation has been carried out the same way as before by modifying the network to initialise with

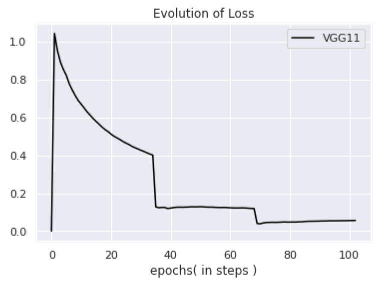
Figure 6. Block Diagram of VGG



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

zero weights and importing a pre-trained model from the PyTorch library [15]. The output classification layer was modified to give 6 outputs instead of the original 1000. Here the hyperparameters were set the same as the main model except for batch size(32). The constraints were due to the hardware limitations and also because we are using a pre-trained model, i.e, we can get a conclusive result in a relatively small amount of time. The evolution of loss curve, training accuracy and F1 score plots can be seen in Fig.7 and Fig.8.

Figure 7. Block Diagram of VGG

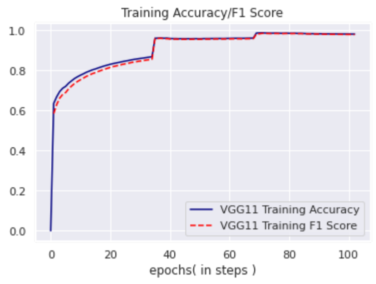


The training accuracy, validation accuracy and validation F1 score per epoch can be observed in table 3, while Testing Accuracy and Testing F1 score is tabulated in table 4.

Epoch	Training	Validation	Validation F1
1	0.869150	0.916166	0.917833
2	0.961310	0.936748	0.936846
3	0.981175	0.941489	0.940508

Table 3. Data in each epoch(Baseline)

Figure 8. Block Diagram of VGG

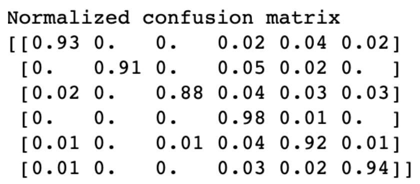


Testing Accuracy	Testing F1 Score
0.93617	0.93601

Table 4. Baseline Model Testing Data

From direct comparison with table 1, we can observe that even though VGG11 offers a slightly higher training accuracy of **1.4%** it offers a decrease in validation score of about **1.2%**. This translates to a even more significant difference when comparing testing accuracy values with table 2, with a major **3%** difference. The normalised confusion matrix can be observed in Fig.9.

Figure 9. Confusion Matrix of VGG



```
Normalized confusion matrix
[[0.93 0.   0.   0.02 0.04 0.02]
 [0.   0.91 0.   0.05 0.02 0.  ]
 [0.02 0.   0.88 0.04 0.03 0.03]
 [0.   0.   0.   0.98 0.01 0.  ]
 [0.01 0.   0.01 0.04 0.92 0.01]
 [0.01 0.   0.   0.03 0.02 0.94]]
```

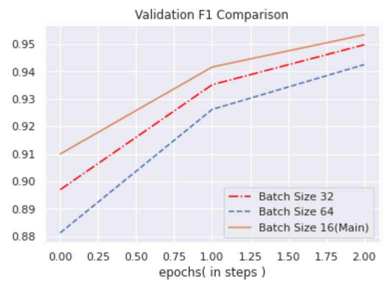
The observed results are expected as DenseNet is a network of high density and more complicated than VGG11 which is an improvement of AlexNet [1] [15].

4. Task 3: Ablation Study

For the next task in the assignment, we will perform an ablation study of our main model from section 2. The 3 hyper-parameters we have chosen for the ablation tasks are **batch size , learning rate** and **epochs**, in that order respectively. For each component/ hyper-parameter we will keep the remaining two hyperparameters fixed and run the models. Keeping in mind our model from section 2, for batch size variation we performed runs for batch size **32 and 64** with learning rate of **2e-5** and **3** epochs. Plots are made comparing validation F1 score for steps-in-epoch for every variation, which is our most crucial performance metric, Fig.10.

We can clearly observe in the fig. mentioned before

Figure 10. Batch Size Variation



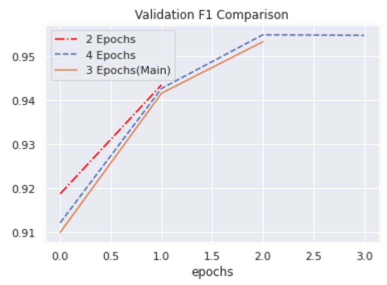
batch size 16 gives the best result. The relevant values including testing accuracy and testing F1 score per batch size are as follows, table 5 :

Batch Size	Testing Accuracy	Testing F1 Score
16	0.95999	0.96239
32	0.94727	0.94953
64	0.93432	0.93748

Table 5. Batch Size Variation Comparison

Next we performed variations on the epochs, i.e, we ran a model at 2 & 4 epochs respectively with batch size 16 and learning rate 2e-5. The following results were obtained and can be seen in fig.11 .

Figure 11. Epoch Variation



From the plot we can clearly observe that even with an extra epoch validation F1 score is similar for the model with 3 epochs, whereas model with 2 epochs has too low of the F1 score. This implies that our model stabilizes on 3rd epoch. Further testing accuracy and testing F1 score can be seen tabulated in table 6 .

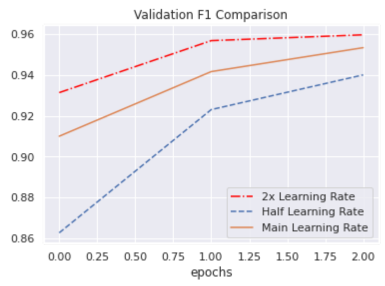
For the last component which is learning rate, we ran two different models again with double and half the values of our main model, i.e, **2e-5 * 2** and **2e-5 / 2** respectively. The obtained validation F1 scores can be seen plotted in the following fig.12.

From the plot we can observe that as soon as we double the learning rate, performance of the model improves very slight compared to our main model by a value of **0.05%**. From this we can infer that our model had already stabilised and learnt at original settings. Further testing values can be seen below in table 7.

Epochs	Testing Accuracy	Testing F1 Score
2	0.93894	0.94309
3	0.95999	0.96239
4	0.95259	0.95485

Table 6. Epoch Variation Comparison

Figure 12. Learning Rate Variation



Learning Rate	Testing Accuracy	Testing F1 Score
2e-10	0.95895	0.95960
2e-5	0.95999	0.96239
2e-2.5	0.93732	0.93936

Table 7. Learning Rate Variation Comparison

From the ablation study we can analyze and confirm that our DenseNet model performs best on setting in section 2. Ablation study gave us an insight on which hyperparameters were most effective and which parameters show a scope of improvement.

5. Task 4: Production Data

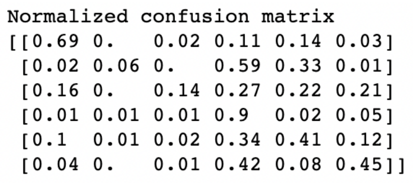
For our final task, we will perform testing on a separately provided production dataset. Our original dataset is Aff-Wild2 dataset [3,5–10,12,13,16] generated using techniques [4,5,11,14]. Before running the production testing we will align the faces like the original dataset. The metrics we have chosen for the evaluation are the same as in previous sections, testing accuracy, testing F1 score and a normalised confusion matrix. The following results were obtained in table 8 and fig.13.

We can observe from the testing results that the provided production dataset performs significantly worse than the original training dataset. One of the possible reasons we can think of is the quality of the provided dataset. We ob-

Testing Accuracy	Testing F1 Score
0.43978	0.35125

Table 8. Production Data Testing

Figure 13. Production Dataset Normalised Matrix



```
Normalized confusion matrix
[[0.69 0.   0.02 0.11 0.14 0.03]
 [0.02 0.06 0.   0.59 0.33 0.01]
 [0.16 0.   0.14 0.27 0.22 0.21]
 [0.01 0.01 0.01 0.9  0.02 0.05]
 [0.1  0.01 0.02 0.34 0.41 0.12]
 [0.04 0.   0.01 0.42 0.08 0.45]]
```

served from the provided set that a tremendous number of faces were blocked by glasses and had reflections on them. This prevented the model to learn by extracting features. In the entirety of it all, our model was fed a completely new kind of data and therefore our accuracy dropped. The technical term for this behaviour of our model is called **Data Drift.**It may take some effort to manage data drift, depending on the nature, extent, and type of the drift. Retraining the model might be enough to manage data drift in this case, but sometimes it may be necessary to start over.

Another reason is that facial expressions can't be just classified into just six expressions. Human facial expressions are much more complex and need a lot of processing and data point extraction for further sub-classification. The model was fed new data with different features than what it is trained for. Thus we can observe such a significant drop in performance. It is in line with actual production models.

A way to improve the performance of the model can be by training the model on a much larger dataset consisting of more diverse faces and expressions. The complexity of the model can be increased to extract features even in an image with blocked faces such as glasses, scarves, reflections, etc. At the same time, it will increase the computational power required to train such a model by many folds. Continuous training data feeding can also be done to prevent the model from degrading over time. In conclusion, our model has performed akin to an actual production model, we have shown in detail analysis with relevant results, inferences and scope of improvements.

References

[1] Sonali Gupta. Vgg-11 architecture, May 2020. 1, 2, 3

[2] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1, 2

[3] Dimitrios Kollias. Abaw: Valence-arousal estimation, expression recognition, action unit detection & multi-task learning challenges. *arXiv preprint arXiv:2202.10659*, 2022. 1, 4

[4] Dimitrios Kollias, Shiyang Cheng, Maja Pantic, and Stefanos Zafeiriou. Photorealistic facial synthesis in the dimensional affect space. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. 4

[5] Dimitrios Kollias, Shiyang Cheng, Evangelos Ververas, Irene Kotsia, and Stefanos Zafeiriou. Deep neural network augmentation: Generating faces for affect analysis. *International Journal of Computer Vision*, 128(5):1455–1484, 2020. 1, 4

[6] D Kollias, A Schulc, E Hajiyev, and S Zafeiriou. Analysing affective behavior in the first abaw 2020 competition. In *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)(FG)*, pages 794–800. 1, 4

[7] Dimitrios Kollias, Viktoriia Sharmanska, and Stefanos Zafeiriou. Face behavior a la carte: Expressions, affect and action units in a single network. *arXiv preprint arXiv:1910.11111*, 2019. 1, 4

[8] Dimitrios Kollias, Viktoriia Sharmanska, and Stefanos Zafeiriou. Distribution matching for heterogeneous multitask learning: a large-scale face study. *arXiv preprint arXiv:2105.03790*, 2021. 1, 4

[9] Dimitrios Kollias, Panagiotis Tzirakis, Mihalis A Nicolaou, Athanasios Papaioannou, Guoying Zhao, Björn Schuller, Irene Kotsia, and Stefanos Zafeiriou. Deep affect prediction in-the-wild: Aff-wild database and challenge, deep architectures, and beyond. *International Journal of Computer Vision*, pages 1–23, 2019. 1, 4

[10] Dimitrios Kollias and Stefanos Zafeiriou. Expression, affect, action unit recognition: Aff-wild2, multi-task learning and arcface. *arXiv preprint arXiv:1910.04855*, 2019. 1, 4

[11] Dimitrios Kollias and Stefanos Zafeiriou. Va-stargan: Continuous affect generation. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 227–238. Springer, 2020. 4

[12] Dimitrios Kollias and Stefanos Zafeiriou. Affect analysis in-the-wild: Valence-arousal, expressions, action units and a unified framework. *arXiv preprint arXiv:2103.15792*, 2021. 1, 4

[13] Dimitrios Kollias and Stefanos Zafeiriou. Analysing affective behavior in the second abaw2 competition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3652–3660, 2021. 1, 4

[14] Andreas Psaroudakis and Dimitrios Kollias. Mixaugment & mixup: Augmentation methods for facial expression recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2367–2375, 2022. 4

[15] PyTorch. Models and pre-trained weights. 1, 2, 3

[16] Stefanos Zafeiriou, Dimitrios Kollias, Mihalis A Nicolaou, Athanasios Papaioannou, Guoying Zhao, and Irene Kotsia. Aff-wild: Valence and arousal 'in-the-wild'challenge. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1980–1987. IEEE, 2017. 1, 4

[17] Chaoning Zhang, Philipp Benz, Dawit Mureja Argaw, Seokju Lee, Junsik Kim, Francois Rameau, Jean-Charles Bazin, and In So Kweon. Resnet or densenet? introducing dense shortcuts to resnet. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3550–3559, 2021. 1, 2