EEEM066 Fundamentals of Machine Learning

Coursework Assignment
Knife classification in real-world images

Antoine EDY, URN: 6797948, ae01116@surrey.ac.uk

Abstract

This work investigates the development of a deep neural network for knife categorization based on photographic input, aiming to aid law enforcement in addressing the surge in knife-related crimes. The study is divided into three key aspects: an exploration of hyperparameters, an analysis of different deep model architectures and a combination of them to achieve the best precision possible. A final model will be proposed in section 3.3, and gives satisfactory results in terms of the precision/computation power balance. Finally, tools for AI explainability will be used to unpack the black boxes that are these models. The NVIDIA®V100 tensor core GPU [11] will be used for the computations, provided by Google Colab Pro.

1. Introduction

1.1. The aim of the coursework

This work aims at classifying knives based on real-life photos. In contrary to other classification problems, the differences between two classes are often very subtle. Therefore, this classification problem requires a Fine-Grained Image Analysis [16].

1.2. The dataset

The image dataset available contains around 10,000 images of knives, divided into 192 classes. This dataset has been enlarged using augmentation, notably rotations, cropping, saturation and brightness tweaking. We divide this dataset into a train and a test dataset, of respectively 9928 and 351 images.

1.3. The strategy used

In order to obtain satisfying results while training a deep neural network, both the architecture and the hyperparameters are to be chosen with the greatest consideration, and require a quantity of experimentations. This work is then divided into two mains parts: section 2 focuses on hyperparameter exploration, specifically examining the effects of learning rates, schedulers, batch size, optimizer and the number of epochs on the model's performance, and section 3 explores different deep model architectures along with variations in their parameters, and two models of Ensemble Learning.

1.4. The evaluation metrics used

To ensure the effectiveness of the training procedure, we should define a loss function. We chose the cross entropy loss [20]:

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log\left(p_i\right)$$

for n classes, where t_i is the truth label and p_i is the Softmax probability for the i^{th} class. This metric is not an evaluation metric, but is still interesting to analyse how fast the model is adapting to the training data.

To evaluate a model, we will use the Mean Average Precision at 5 (MAP@5) [19] on a separate test dataset, which is the classical method of evaluation for machine-learning models. It is calculated using the following formula:

$$\mathrm{MAP@5} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\mathrm{TP}_i}{i} \times \mathrm{rel}_i\right)$$

where N is the total number of examples, TP_i is the number of true positive predictions for the i-th example in the ranked list of predictions (up to rank 5) and rel_i is an indicator function that is 1 if the i-th example is relevant (correctly predicted), and 0 otherwise.

This formula represents the average precision at each rank, and the mean is taken over all examples. We will use both MAP@5 and MAP@1 to compare the performances of the different models during the coursework.

2. Exploration of the hyperparameters

While training a neural network to accomplish any task, the choice of the hyperparameter is crucial and requires some experimentations. To explore the effect of the main ones, we will, in this first section, be using the proposed architecture model EfficientNet-B0 [10].

2.1. The effect of the learning rate

The learning rate parameter determines the size of the step taken during the optimization process: it influences how much the model's parameters are updated with each iteration of the training process. Choosing an appropriate learning rate is crucial for training a deep neural network effectively. Figure 1 shows the experimentations done to find the optimal value of the learning-rate for our specific task.

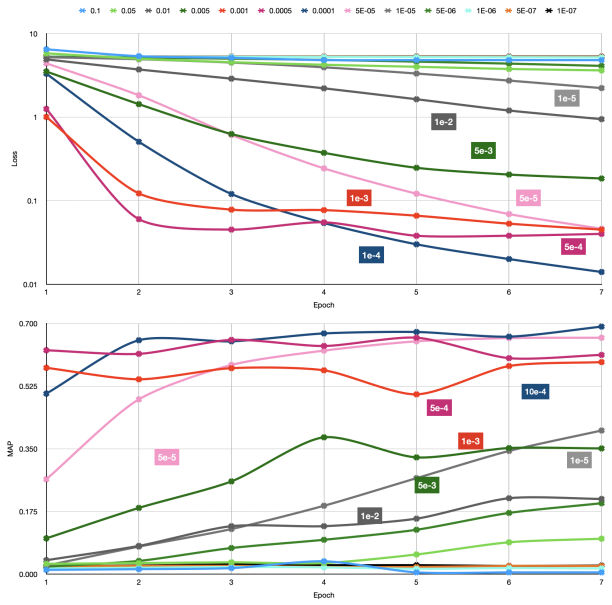


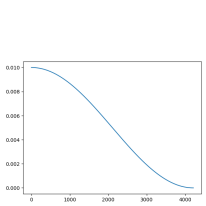
Figure 1. Variation of the loss (top, logarithmic scale, lower is better) and the mean average precision (bottom, higher is better) through the number of epochs and for different learning rates

A very high learning rate ($lr = 10^{-2}$) fails to give satisfactory results, because it exceeds the value aimed at each iteration of the training process. A high learning rate ($lr = 10^{-3}$ or $lr = 5*10^{-2}$) causes the model to converge quickly. However, it overshoots the minimum and lead to a slow convergence to zero loss. On the other hand, a low learning rate ($lr = 10^{-5}$) causes a very slow convergence. The optimal learning rate here seems to be around $lr = 10^{-4}$, the dark blue curve on Figure 1. After 7 epochs only, we reach a loss value of 0.014 and a satisfying MAP@5 of 69.1%.

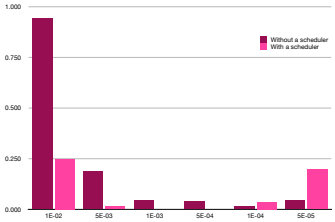
2.2. The effect of the scheduler

However, it seems preferable to have a high learning rate at the beginning of the training process in order to make the parameters of the neural network converge quickly, then to lower it down to fine-tune these parameters and increase accuracy. This concept is called scheduling, and the tool used is a scheduler.

We plot the value of the learning rate through time during the training on figure 2a. We will use the Cosine Annealing scheduler: with an initial learning rate of 10^{-2}, its value goes down to almost zero, where the fine-tuning operates.



(a) Learning rate through time during the training of the network (Cosine Annealing)



(b) Comparison of the loss of the model with different initial learning rates after 7 epochs (lower is better)

Figure 2. Using a Cosine Annealing scheduler

Figure 2b compares the loss of the model after 7 epochs of training, with and without the use of a scheduler. We can notice that the use of a scheduler offers an important increase in performance, and specifically when the initial learning rate is high, as explained at the beginning of 2.2. If the initial learning rate is too low, lowering it even more during the training is not desirable.

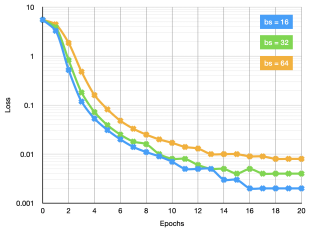
Using the scheduler, we find the best result after 7 epochs only with a learning rate of $1 * 10^{-4}$, with a precision of 70.4% and a loss value of 0.003.

2.3. The effect of the batch size

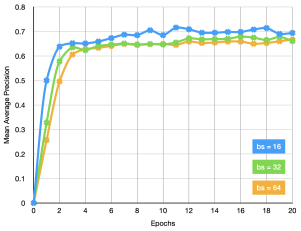
For this next section, we will use the optimal hyperparameters we found in section 2.2, so an initial learning rate of $1*10^{-4}$ and the use of a scheduler. We will vary the batch size, and test with $batch\_size \in \{16, 32, 64\}$. The batch size defines the number of samples to work through before updating the internal model parameters in a deep neural network. In general, large batch sizes leads to faster training times but may result in lower accuracy and overfitting, while smaller batch sizes can provide better accuracy, but can be computationally expensive and time-consuming. This general result is verified by the experimental results showed in table 1 and on figure 3.

Batch size	loss	mAP	time (min)
16	0.001	0.741	30
32	0.004	0.661	20
64	0.008	0.667	18

Table 1. Effect of the batch size after 20 epochs



(a) Loss through the epochs



(b) mAP through the epochs

Figure 3. Loss and mAP curve for different batch sizes

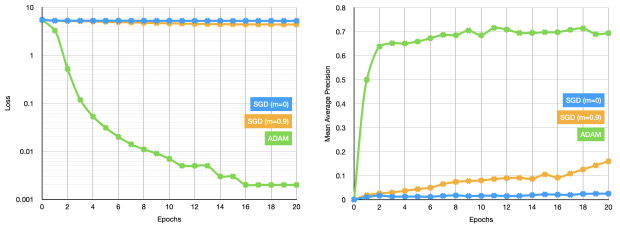
After seeing these results, we might opt for a small batch size to optimize the precision of the model, but we must keep in mind that this choice is computationally expensive and time-consuming.

2.4. Comparing the optimizers

In this section, we will compare the results using the Adam Optimizer [9] and the SGD Optimizer with different momentum [13]. While Adam uses adaptive learning rates for each parameter, with the SGD Optimizer, the learning rate is equal for all of them. Without a momentum, the update is solely based on the current gradient, without considering past gradients. The variant of SGD that includes momentum takes into account the past gradients in updating the parameters. We will call the momentum γ as it is usually done in the literature.

Optimizer	loss	mAP (%)
SGD ($\gamma = 0$)	5.226	2.5
SGD ($\gamma = 0.9$)	4.398	16.0
Adam	0.002	69.4

Table 2. Effect of the optimizer after 20 epochs



(a) Loss through the epochs (b) mAP through the epochs

Figure 4. Loss and mAP curve for different optimizers

The results are indisputable, and while adding a momentum gives a significant boost in performances to the SGD optimizer, Adam gives remarkably better results. In fact, SGD optimizers are outdated nowadays, and Adam is, according to the literature, almost always the way to go while training deep neural networks [2].

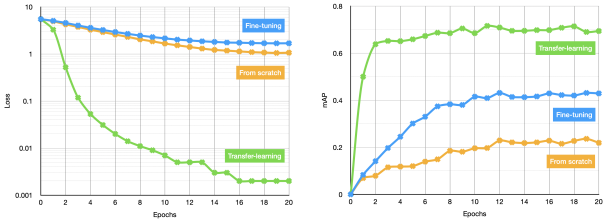
2.5. Fine-Tuning vs. Transfer learning vs. Learning from scratch

Fine-tuning, transfer learning, and learning from scratch are three different approaches to training neural networks, each with its own advantages and use cases. Again with the EfficientNet-B0 architecture, let's explore each of them:

- Learning from Scratch: the model starts with random weights, and the entire network is trained on a specific task.

- Transfer learning involves using a pre-trained model on a source task and fine-tuning it for a different but related target task. The idea is to leverage the knowledge gained by the model on the source task and apply it to the target task.

- Fine-tuning is a specific form of transfer learning in which, very often, some layers are frozen during the training. By training only the last layers to a specific task, we make use of the semantic feature extracted from the original network.



(a) Loss through the epochs (b) mAP through the epochs

Figure 5. Loss and mAP curve for different training techniques

If training from scratch is so inefficient in our case, it is because we don't have enough computation resource to fully train the model. Moreover, the pre-trained models we are using, EfficientNet-B0, has shown incredible results in image classification tasks [10]: it is no surprise that transfer-learning is efficient. Finally, we notice that freezing the first layers do not improve drastically the precision in our case. There is a significant difference between the data distribution of the pre-training task and the fine-tuning task (the model has been trained on ImageNet-1k [6], and our knife images are quite different). Indeed, pre-trained models may have already learned representations that are optimal for the source task, and freezing the initial layers prevents these layers from adapting to the nuances of the target task.

3. Deep model architectures

The choice of a deep neural network architecture is crucial as it defines the model's capacity to learn complex patterns, influences computational efficiency for both training and inference, and determines its adaptability to specific tasks, impacting overall performance and generalization.

3.1. Using different networks

In this section, we will analyse and compare some of the most popular CNN architectures : EfficientNet [10] (the model we were using from the beginning of the coursework), ResNet-18 [8], MobileNet v.3 [1] and SKResNet [18]. These DNN models exhibit a modest parameter count, with the most efficient ones nowadays possessing at least four times as many parameters. These were chosen to match the computation power available for this work.

3.1.1 The specificities of the architectures

EfficientNet uses compound scaling [10]. The general idea is that the best way to scale a model is to do it in the three dimensions: width, depth, and image resolution.

ResNet tackles the vanishing gradient problem, an issue

that occurs in very deep neural networks (the gradients employed for updating the network diminish significantly as they propagate backward from the output layers to the earlier layers during backpropagation). A skip connection allows the signal to bypass one or more layers and to be added directly to the output of the network. [8]

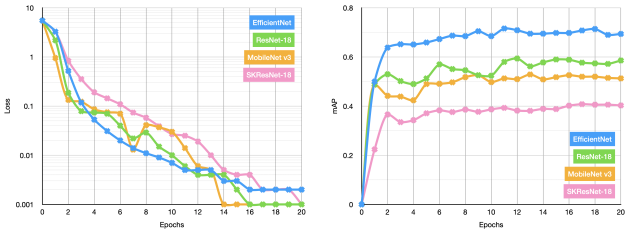
MobileNet minimizes the number of parameters, by using Depthwise Separable Convolution and Pointwise Convolution [12].

SKResNet builds upon the ResNet architecture by incorporating the concept of receptive field sizes, resulting in the inclusion of Selective Kernel units [18].

3.1.2 A comparison of deep CNN architectures

Architecture	Number of param.	Loss	mAP
EfficientNet-B0	11M	0.002	0.694
ResNet-18	11M	0.001	0.586
MobileNet	13M	0.000	0.513
SKResNet-18	12M	0.001	0.403

Table 3. Comparison of different CNN architectures after 20 epochs of training



(a) Loss through the epochs (b) mAP through the epochs

Figure 6. Loss and mAP curve for different model architectures

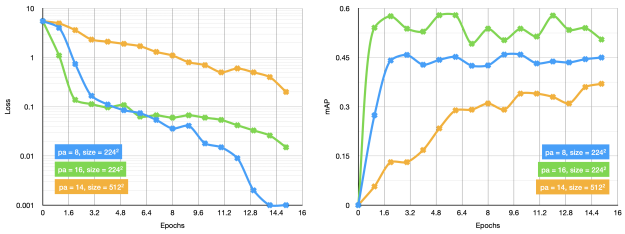
As shown again with these results, EfficientNet models have consistently achieved state-of-the-art results on various benchmark datasets. Moreover, its design principles prioritize computation efficiency. Finally, compound scaling aims to adapt to specific characteristics of different tasks [10]. Thus, transfer learning is very efficient. On the contrary, ResNet's skip connections may lead to overfitting during fine-tuning, while MobileNet's lack of regularization hinders effective generalization, especially with a small target dataset.

The largest model trained for the purpose of this coursework is EfficientNet-B3, with 78.6% of MAP@5 after 55 epochs (5 hours training). The log file can be found attached in `log_effnet-b3.txt`.

3.2. Comparison of Vision Transformer (ViT) image classification model

Along with CNNs, Vision Transformers stands out as one of the foremost choice for image classification tasks. Unlike CNNs, ViTs process images using self-attention mechanisms that allow them to consider relationships between all image regions simultaneously, enabling them to capture a more global context.

Two new hyperparameters are to be chosen when using a ViT: the patch size and the size of the input image. The patch size informs a speed/accuracy tradeoff (smaller patches have higher accuracy and compute cost). Indeed, the idea behind ViT is to separate input images into non-overlapping patches and conduct computations on tokens from these patches. We proceed to computation using three different models [15] [4], [3], and the results are shown on 7. The models are imported from the timm [17] library, hosted on Hugging Face [5].



(a) Loss through the epochs (b) mAP through the epochs

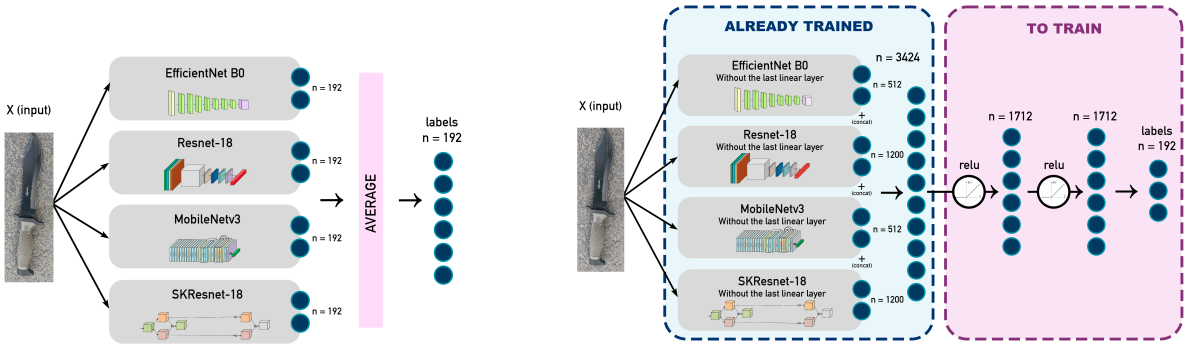
Figure 7. Loss and mAP curve for different ViT models. `pa` refers to the patch size, and `size` is the input image size.

ViTs exhibit inferior performance compared to earlier CNNs, which might be because they often need more training data than CNNs. Additionally, because CNNs can be parallelized, they are computationally efficient and better suitable for real-time and resource-constrained applications. Therefore, we won't be using ViTs, which have shown to be less efficient in our case (MAP@5 $< 60\%$).

3.3. Combining the models

Combining the results of different models, often referred to as ensemble learning, can be beneficial to reduce the overfitting, to increase the robustness to noisy data and to make the most of complementary architectures. In fact, models with different architectures capture different aspects of the underlying patterns in the data. Combining them can exploit their complementary strengths.

Figure 8 illustrates the two techniques I implemented to combine the models. The first one (figure 8a) is an Ensemble learning technique, in which each of the scores attributed to each of the knife classes are averaged. The second one (figure 8b) is more complex and offers better results (table 4). In this latter technique, we take the output of the penultimate layer of different models, and use that as inputs of a shallow neural network that we can train. In a CNN, the last few layers typically contain high-level features or representations of the input images, and this is crucial in our task where understanding the overall structure of an image is essential (to discriminate the background for example).



(a) Ensemble learning

(b) New model

Figure 8. Two different approaches

Architecture	MAP@1 (%)
EfficientNet-B0	59.4
ResNet-18	60.9
MobileNet	68.8
SKResNet	45.3
Ensemble learning (8a)	71.9
Shallow NN (8b)	73.4

Table 4. Results of two model combination techniques

The result of 4 is satisfactory in the sense that the new models score a better precision than the best of the others. Intuitively, the Ensemble learning method where we average the labels works best because it reduces overfitting and compensate the errors of each model. DNN models may specialize in learning specific features: combining their outputs allows the shallow NN to leverage complementary information, enhancing the overall model's performance.

4. Explainability

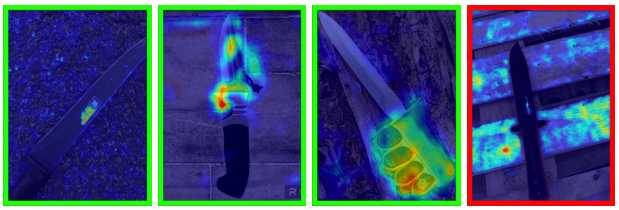
AI explainability is crucial for understanding and interpreting complex models known as *black boxes*, fostering transparency, uncovering potential biases, and ensuring responsible and accountable deployment of advanced machine learning systems in real-world applications. As a bonus part of this coursework, we will explore Grad-Cam [14], one of the most used explainability tools for image classification and object recognition.

Grad-CAM, which stands for Gradient-weighted Class Activation Mapping, is a technique used for visualizing the regions of an input image that are important for making a particular prediction by a CNN. The technical aspect of Grad-CAM can be found in the following paper [14].

The algorithm generates a heatmap, which highlights the regions of the input image that contributed the most to the prediction of the target class. It provides a way to interpret and understand the decisions made by deep neural networks. By visualizing the areas of an image that influence the model's decision, researchers and practitioners can gain insights into potential model errors or biases.

We will use this `Github` repository to compute the Grad-Cam algorithm on some of our models and input images [7].

We will explore a ResNet-18 model trained with our dataset.



(a) (b) (c) (d)

Figure 9. Examples of Grad-Cam on test image (ResNet-18)

We can notice on figures 9a, 9b and 9c that Grad-Cam successfully retrieve the parts of the images that are the most relevant to classify them. In order: the text on the knife, the original clasp and the unique handle. Moreover, Grad-Cam also emphasizes where the model fails. On figure 9d, we notice that the model looks at the background of the image, whereas the background of an image should not be relevant when doing knife classification. Thus, it leads to the model failing to retrieve the correct class of the knife.

5. Conclusion

In addition to studying the key effects of hyperparameters and the main architecture of deep neural networks, this study underscores the notable effectiveness of ensemble learning techniques. We found that combining diverse models significantly boosts predictive performance (as seen in table 4) and requires no to minimal additional training. This efficiency makes ensemble learning an attractive approach, delivering gains without extensive retraining.

Looking ahead, the importance of AI explainability tools grows in shaping the future of artificial intelligence. With increasingly complex models integral to decision-making, understanding their decisions is crucial. Tools like Grad-CAM play a pivotal role in demystifying complex models, ensuring transparency for trust and uncovering potential biases or errors. As we chart AI development's trajectory, robust explainability mechanisms become a cornerstone, ensuring responsible deployment of advanced machine learning models in real-world applications.

References

[1] Grace Chu Liang-Chieh Chen Bo Chen Mingxing Tan Weijun Wang Yukun Zhu Ruoming Pang Vijay Vasudevan Quoc V. Le Hartwig Adam Andrew Howard, Mark Sandler. Searching for mobilenetv3. *Google AI, Google Brain*, 2019. 3

[2] Jason Brownlee. Machine learning mastery: Gentle introduction to the adam optimization algorithm for deep learning. 3

[3] Mathilde Caron, Hugo Touvron, Ishan Misra, Herv'e J'egou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 4

[4] Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2818–2829, 2023. 4

[5] Thomas Wolf (CSO) Clément Delangue (CEO), Julien Chaumond (CTO). Hugging face. 4

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 3

[7] Jacob Gildenblat and contributors. Pytorch library for cam methods. https://github.com/jacobgil/pytorch-grad-cam, 2021. 5

[8] Shaoqing Ren Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. *Microsoft Research*, 2015. 3, 4

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 3

[10] Quoc V. Le Mingxing Tan. Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*, 2019. 1, 3, 4

[11] NVIDIA®. V100 gpu. https://www.nvidia.com/en-gb/data-center/v100/. 1

[12] Abhijeet Pujara. Image classification with mobilenet, 2023. 4

[13] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. 3

[14] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct. 2019. 5

[15] Andreas Steiner, Alexander Kolesnikov, , Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021. 4

[16] X. Wei, Y. Song, O. Aodha, J. Wu, Y. Peng, J. Tang, J. Yang, and S. Belongie. Fine-grained image analysis with deep learning: A survey. *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, 44(12):8927–8948, dec 2022. 1

[17] Ross Wightman. Pytorch image models. https://github.com/huggingface/pytorch-image-models, 2019. 4

[18] Xiaolin Hu Jian Yang Xiang Li, Wenhai Wang. Selective kernel networks, 2019. 3, 4

[19] Ethan Zhang and Yi Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009. 1

[20] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels, 2018. 1