Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Plzeň, 2023 Bc. Tomáš Majer

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Tomáš MAJER**
Osobní číslo:	**A21N0116P**
Studijní program:	**N3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Téma práce:	**Odhad pózy zvířat pomocí metod hlubokého učení**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Seznamte se s problematikou odhadu pózy zvířat se zaměřením se na metody hlubokého učení.
2. Popište existující databáze určené k odhadu pózy zvířat.
3. Pomocí knihovny MMPose dotrénujte vhodné předtrénované modely na interní databázi rysů.
4. Vyhodnoďte a porovnejte výsledky všech testovaných metod.

Rozsah diplomové práce:	**40-50 stránek A4**
Rozsah grafických prací:	
Forma zpracování diplomové práce:	**tištěná**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

- Jiang, Le, et al. „Animal pose estimation: A closer look at the state-of-the-art, existing gaps and opportunities.“ *Computer Vision and Image Understanding* (2022): 103483.
- https://mmpose.readthedocs.io/en/latest/tasks/2d_animal_keypoint.html

Vedoucí diplomové práce:	**Ing. Marek Hrúz, Ph.D.** Katedra kybernetiky
Datum zadání diplomové práce:	**1. října 2022**
Termín odevzdání diplomové práce:	**22. května 2023**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

L.S.

[illegible]

Doc. Ing. Miloš Železný, Ph.D.
děkan

[illegible]

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 1. října 2022

Poděkování

Nejdříve bych rád poděkoval vedoucímu své práce Ing. Marku Hrúzovi, Ph.D. za čas věnovaný konzultacím, vstřícnost a za všechny cenné rady, které mi dal. V druhé řadě bych rád poděkoval své rodině, přátelům a svým spolužákům za podporu během studia.

Anotace

Cílem práce je studium, použití a vyhodnocení metod řešení úlohy estimace pózy zvířat. Tyto metody jsou použity na novém Lynx-Pose datasetu, který je v práci představen.

Práce sestává z úvodu do teoretického základu estimace pózy a použití metod odhadu pózy k vyřešení predikce pózy rysa ostrovida. V práci jsou představeny hluboké neuronové sítě a metody vyhodnocení pro estimaci pózy spolu s dostupnými datasety pro estimaci pózy zvířat. Experimenty sestávají z natrénování modelů HRNet-W32, ResNet-50 a ResNet-152 pro estimaci pózy rysů a z použití technik augmentace obrazů k dosažení lepších predikcí pózy. Výsledky jsou vyhodnoceny za použití mean average precision a procenta správných predikcí a vykresleny na obrázcích z Lynx-Pose datasetu.

Klíčová slova: estimace pózy zvířat, 2D estimace pózy, estimace pózy nerigidních objektů, estimace pózy rysa, HRNet, ResNet, augmentace obrázků, Lynx-Pose dataset, dataset pro estimaci pózy zvířat

Annotation

The goal of the thesis is to study, use and evaluate the animal pose estimation methods. These methods are to be used on the novel Lynx-Pose dataset introduced in the thesis.

The thesis consists of an introduction to the animal pose estimation theoretical basis and the use of the pose estimation methods to predict the Eurasian lynx pose. Deep neural networks and evaluation metrics for pose estimation are introduced as well as the available animal pose estimation datasets. The experiments consist of training the HRNet-W32, ResNet-50, and ResNet-152 models for lynx pose estimation and using image augmentation techniques to achieve better performance. The results are evaluated using the mean average precision and the percentage of correct predictions and also visualized on images from the Lynx-Pose dataset.

Keywords: animal pose estimation, 2D pose estimation, non-rigid object pose estimation, lynx pose estimation, HRNet, ResNet, image augmentation, Lynx-Pose dataset, animal pose estimation dataset

Contents

1 Introduction

In recent years, society has become increasingly aware of the nature protection importance. Academia follows this trend and some of the recent research shifts more towards sustainability and wildlife conservation. As we see in the wildlife conservation works, one of the important used techniques to collect data is wildlife tracking [25, 32, 33, 58]. Wildlife tracking studies the movement and movement patterns of animal individuals or whole populations. The movement is examined both from a local and a global point of view. An example of a local, respectively global point of view is the movement of different animal species on Barro Colorado island [33], respectively the mule deer across the continent [58] or even migration of birds across multiple continents [32].

My thesis is part of a work with the goal of wildlife tracking. Specifically, the goal is tracking wild Eurasian lynxes in the Šumava and Bayerwald National Parks. Let us now look at why wildlife tracking is important for wildlife conservation.

There are multiple wildlife tracking uses important for wildlife conservation. One of the uses is the detection and analysis of issues with the animals' habitat. The tracked wildlife often migrates within their habitat to find new food sources. This can be affected by the creation and extension of the built environment. Migration is vital for some animals and it can be destructive for a given population if an important part of their migration route is blocked [58]. Animals can also change their migration patterns on the basis of climate changes, which can in turn cause a local abundance of a given species in the area the animals migrate to [59]. Scientists also use wildlife tracking to research the land use change impact which can cause changes in biodiversity. This can affect the species within a given ecosystem and have an impact on the extinction risk of entire populations [74]. Tracking helps with animal protection too. Wildlife can be endangered by invasive species or by infectious diseases. The diseases and parasites spread and the impact of invasive species can be studied using movement patterns obtained by tracking [33]. Wildlife tracking is also used in poaching prevention and there is an increasing number of works studying this topic [2, 13, 31, 35]. The rangers or nature reserve guards use the knowledge of wildlife movement and position themselves for more effective animal protection. The tracking system can also detect rapid changes in animal movements possibly caused by a panicking animal [2].

We now look at wildlife tracking methods. One of the first wildlife tracking methods is animal tagging. One of the early recorded experiments is from 1834 [1], and this method is still used nowadays. The scientists capture the individual animal and tag it, the tracking is based on recapturing the given individual, retagging it, and recording new information in the process. The drawbacks of this approach are the need to physically recapture the animal and the method's invasiveness.

To moderate these disadvantages, more modern wildlife tracking methods were developed. Some of these methods are still based on the previously mentioned principle - the individual is captured and then given a tag, but nowadays the tag is a digital device or a location transmitter. These devices are called biologgers and some can record environmental conditions or even video. They can be of various natures - a Passive Integrated Transponder (PIT) which is an electronic microchip serving as a digital tag [19], Global Positioning System (GPS) trackers to observe relatively fine-scale movement [10, 14, 28], radio signal transmitters with the radio telemetry use and radio-frequency identification [8, 16, 33], and more. Examples of such biologger devices on animals are shown in Figure 1. The devices used by these methods can be costly, especially when satellite tracking is used [64]. While

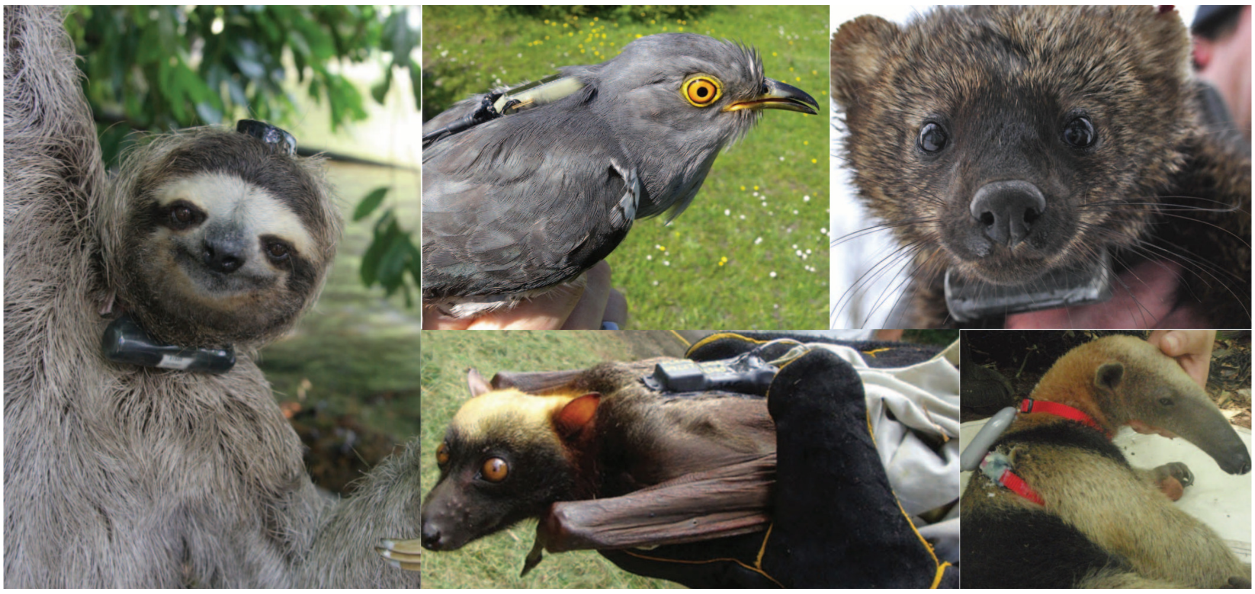


Figure 1: Examples of captured animals with biologging devices [32].

the price decreases as the device development is progressing, there is still the disadvantage of the methods' invasiveness.

All of the previously mentioned methods require capturing the animal in order for a biologger to be attached to it. Since the devices are installed directly on the animals, an effort is being made to keep their size small. The device has to be carried and its weight and size must therefore not exhaust the animal, hinder its movement or reduce its appeal to potential mates. In Figure 1 we see that even with the more modern devices the size is still big and the device is definitely a burden for the animal. The size of the devices is also limited by the battery size. Ideally, the biologger should also be able to transmit the data remotely to prevent unnecessary animal recapturing. This remote transfer contradicts the size requirement because the remote transfer often requires bigger devices with stronger batteries [69].

Some of the newer methods take these disadvantages into account and base themselves on non-invasive observation of individuals or populations [18, 23, 40, 49]. Therefore, these methods do not require capturing the animal and do not disturb the life of the animals. This also means they are often cheaper and the data collection and device replacement are easier.

The prominent method is camera trap wildlife tracking which, as the name suggests, uses the pictures from camera traps to identify and track the individual animal [34]. This identification along with the camera trap location and a timestamp gives us a partial model of the movement and behavior of the given individual.

As mentioned, to track the individual we need to identify it from the image. As already stated, we focus on the Eurasian lynx tracking. Lynxes, like other animals, have unique markings and can be identified by those markings making them ideal animals for camera trap capture-recapture (re-identification) methods [20, 55, 68]. The lynx initial and every subsequent lynx identification are done manually by an expert. However, the identification process could be automated and such automation allows the experts to focus on more important tasks like processing the data from wildlife tracking. To be able to automate the process and identify a lynx individual automatically, we need to extract the markings from the image. This would also allow us to identify a new lynx individual if we extracted an unknown marking pattern.

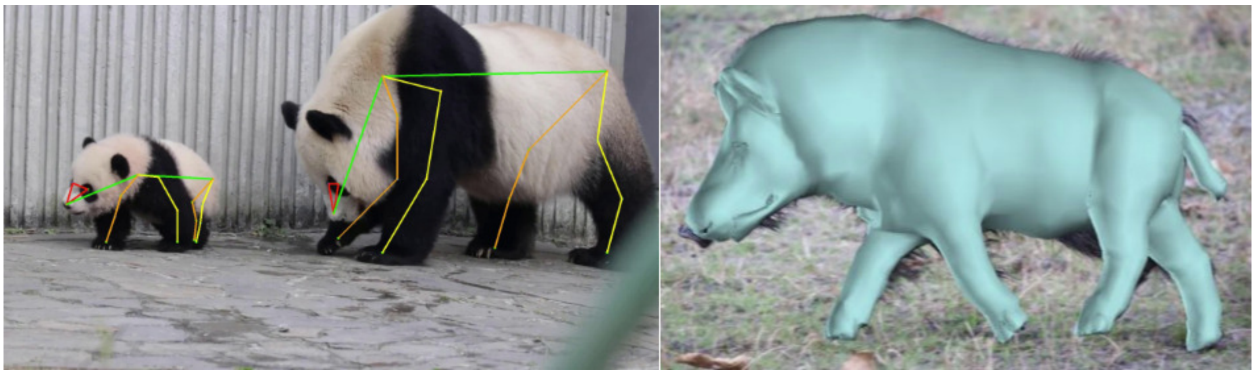


Figure 2: Example of a kinematic (left) and a volumetric (right) animal pose models [71, 77].

Therefore, we need the ability to extract the coat pattern from the image for lynx identification and tracking. One of the possible approaches is to find the lynx's pose and use it to extract the texture from the image. This way we know the correspondence of texture parts to body parts and we can then compare the texture with a set of known geometrically normalized textures. The pose of the lynx can be found with the use of pose estimation.

The pose estimation problem is a general problem consisting of detecting the position and orientation of a person or an object. This definition suggests that the human pose estimation task is a very important subtask of pose estimation. Therefore we first define human pose estimation which we later reformulate into an animal pose estimation definition. Human pose estimation can be defined as a problem of human joints localization in an image [66]. This means we look for points (areas) in the image, which correspond to important human body parts. These important parts can be joints in the skeleton, other points of interest such as the eyes or nose, or even whole limbs. If we look for singular points of interest, they do not always need to correspond to exact joints in the human skeleton. However, they are still often called joints. The estimation can output various pose models which we now introduce briefly.

As stated in [75], the model representing the pose by the 2D or 3D coordinates of singular points is called a kinematic (skeleton-based) model. For 2D pose estimation, we can also use the planar (contour-based) model which replaces body parts with rectangle approximations of the body contours. An example of such a model is the cardboard model [30], and another example of a planar model is the well-known Active Shape Model [12]. The last option, the volumetric model, is used for 3D pose estimation and represents the pose as geometric meshes and shapes, an example of such a model is the Skinned Multi-Person Linear model [43].

Animal pose estimation is derived from human pose estimation as the animal joints localization in the image [29]. We notice that the formal definition is the same for both problems and the principles described above apply to the animal pose estimation task too. The difference is in the task-dependant definition of the joints and the used models. The joints for the animals often correspond to their body parts (ears, muzzle, skeleton joints, paws, tail base, . . .) as well, but we notice that the body parts we listed here differ slightly from the ones used in human pose estimation (e.g. tail base). For 2D and 3D animal pose estimation, kinematic models are often used. The other model types defined above also exist for animal pose estimation tasks. a volumetric model example for 3D animal pose estimation is the Skinned Multi-Animal Linear (SMAL) model [77]. a kinematic and a volumetric pose model examples for animals are depicted in Figure 2.

The rigidness of the studied subject is another aspect of pose estimation. The methods and definitions presented so far all regarded the non-rigid object pose estimation. The methods of rigid object pose estimation are also being studied. They are important for many relevant and trending problems like autonomous driving, robotic manipulation, and augmented reality. With the object being rigid, the pose is estimated in regard to the camera location, meaning the 3D location and 3D rotation of the object are estimated. This results in a 6D vector hence the task is often called 6D pose estimation [26, 44]. These methods go beyond the scope of my thesis as they are not of big importance for tracking lynxes.

To summarize, in my thesis, the focus is on the non-rigid object pose estimation, and the kinematic model is used for our task. We use the kinematic model because as stated in [29], the 2D pose estimation tasks usually aim "to detect the 2D coordinates of the keypoints (joints) of an animal".

Having presented the pose estimation task, the automation method for lynx identification and tracking can be concluded. Given the lynx image from a camera trap, we can estimate the 2D animal pose in the image and extract the coat pattern texture which can be used for the identification of the individual. The animal can then be tracked with the use of the camera trap location and the information about the time animal walked by the camera trap. This way, wildlife tracking can be done without the need to catch the animals. As [23] states, the selected spatial placement of the cameras influences the result greatly so this aspect needs to be considered too.

To further expand on the uses of animal pose estimation beyond my thesis, animal pose estimation can also be used for behavioral analysis [22, 46, 54] or 3D model extraction [4, 38, 77]. Stating all the possible uses of animal pose estimation we notice it is of similar importance as human pose estimation. However, since it does not directly benefit humans, for example from a health, financial, or military point of view, it is less researched and funded. This is related to the lack of data, which is also many times more difficult to obtain in some animal species than in humans because of their population number. The data scarcity problem is not uncommon and has a good solution in computer vision tasks that also helps the methods to generalize - data augmentation [50, 63, 76]. Thus we also explore the data augmentation techniques in the lynx pose estimation task.

In the theoretical part of my thesis, the tasks of animal and lynx pose estimation are introduced in Section 2. Then the existing deep learning methods used to estimate the pose are studied in Section 3. Afterward, methods for pose estimation evaluation are explored in Section 4, our novel dataset for lynx pose estimation is introduced and the existing datasets for which pre-trained models exist are also looked at in Section 5. Afterward, we introduce the image augmentation techniques in Section 6. Section 7 is the practical part of my thesis where the deep learning models are selected and trained on our dataset. We also conduct a study of the data augmentations' impact on the training.

2 Animal Pose Estimation

In this section, we introduce the tasks of pose estimation and animal pose estimation in more detail, and introduce the approaches used for pose estimation depending on the expected input and output.

As mentioned in the Introduction, the non-rigid object pose estimation task is often performed on humans and we also already mentioned the animal pose estimation task. Let us now quickly recap this definition by defining the general pose estimation task. The goal of general non-rigid object pose estimation is to detect the location of a given object's parts. These parts are predetermined and often represented as points in the 2D image or in 3D space. Together, the points give us information about the object configuration (pose) and position.

The animal pose estimation task is a subtask of the general non-rigid object pose estimation task. The object is an animal and the detected parts very often correspond directly to specific animal body parts, such as ears, eyes, limbs, or tail.

As stated in [29], we can also define several different aspects of the pose estimation task depending on the expected input and output. In terms of input, we are primarily interested in whether we are estimating the pose for a single image or for a video, i.e., an image sequence. Another possibility is an input consisting of multiple images representing multiple views in a single moment.

In terms of output, there are several aspects to consider. First, we need to decide on the model we use for the pose as we already stated in the Introduction. As [75] states, we can choose from the kinematic (2D or 3D pose), planar (2D pose), or volumetric (3D pose) model. Let us suppose we use the kinematic model. Using this model we can estimate the pose in 2D or 3D which affects the coordinate dimension we get as the output. We are also interested in whether we always estimate the pose for only one occurrence of a given individual (single animal pose estimation), or whether there may be multiple individuals and we estimate the pose for all of them (multiple animal pose estimation). All of the mentioned needs to be taken into account in general pose estimation tasks. In the animal pose estimation task, we additionally consider whether we are estimating the pose for a single species or for multiple species.

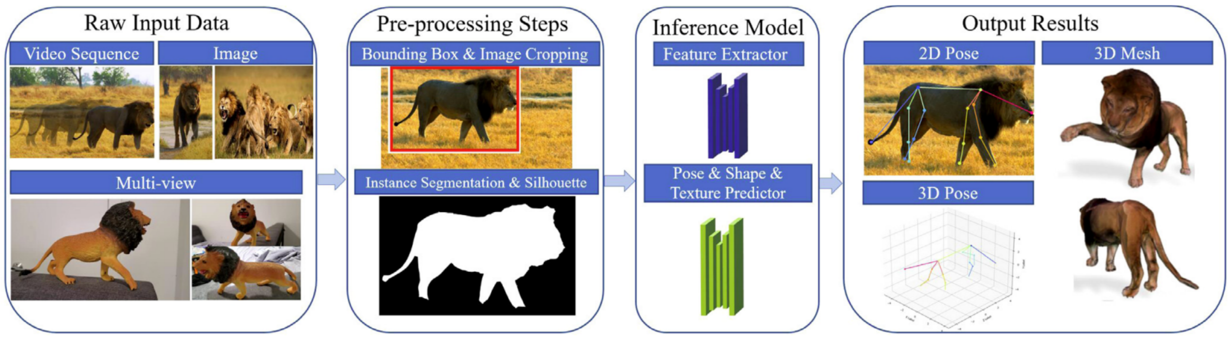


Figure 3: A generic animal pose estimation workflow flowchart [29].

We see the above-mentioned aspects in Figure 3, as well as few other things that need to be considered. The Figure 3 shows the image pre-processing as the inference model receives the content of the object's bounding box as input. This approach is called top-down and requires to have a detector (e.g. Single shot multibox detector [42], R-CNN [21]) between the raw input and the pose estimation model. Examples of backbones used for top-down pose estimation are Residual Networks (ResNets) [24], Cascaded Pyramid Network (CPN) [9], or High-Resolution Network (HRNet) [67]. If the detector detects multiple objects, they are passed into the pose estimation model as individual inputs. The second approach is called bottom-up (e.g. OpenPose [7]). This approach estimates the pose of all objects' parts by itself, groups these parts, and creates objects from them.

2.1 Lynx Pose Estimation

In terms of the presented aspects, our task can be defined as follows. We estimate the pose for a single image on input. We solve a task of 2D animal pose estimation using a kinematic (skeletal-based) pose model and the estimation is done for a single species - The Eurasian lynx (Latin: Lynx lynx).

Since the Eurasian lynx is mostly a solitary animal [62], occurrences of two or more lynxes in a single image are rare. This leads us to the usage of the top-down approach to pose estimation. If there are ever two (or more) lynxes in a single image, we assume that a lynx detector exists and gives us information about all of the lynxes by means of multiple bounding boxes. In this thesis, we work only on the pose estimation part of this issue and assume that the detector is given. This detector is simulated by having the annotated bounding boxes for all the ground truth images.

In our task, we detect the following 20 keypoints: two ears, two eyes, a nose, withers, a neck, a tail base, four paws, four "knees", two shoulder points, and two hip points. In Figure 4, we show an example of a lynx from our dataset and the points that are annotated and represent the ground truth pose.

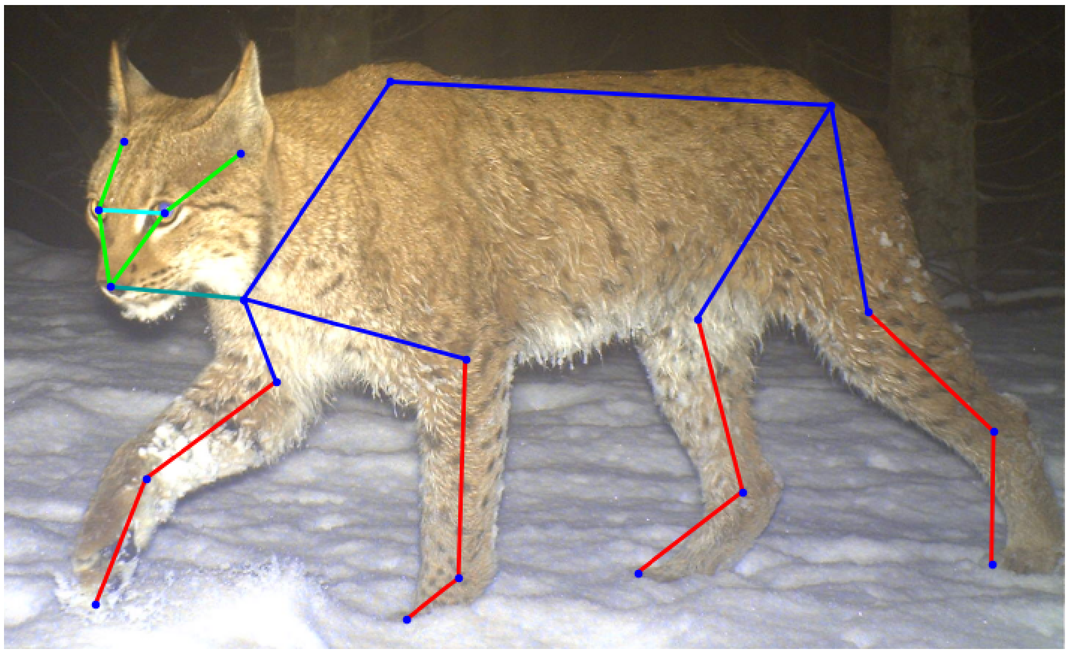


Figure 4: Example of an annotated Eurasian lynx image from our dataset.

We notice that these keypoints correspond to anatomical parts of the lynx skeleton with the only exception being the throat keypoint. The "knee", hip, and shoulder keypoints correspond to physical joints in the lynx skeleton. The ear keypoints are the exact points where the lynx skull ends and the ear cartilage tissue starts. If we connect these keypoints with the eye keypoints on the corresponding side and then connect the eye keypoints with the nose keypoint, a triangle is formed. The line going from the nose to the eye and to the ear should be almost straight and this should be the case in all lynx images. We see that the image in Figure 4 meets this requirement, but the left ear base could have been annotated slightly closer to the eye. The withers keypoint is placed exactly between the shoulder blades of the lynx and the tail base keypoint is between the topmost parts of the pelvis where the lynx's tail starts. The paw keypoints should always be annotated exactly at the very end of the lynx's limbs.

Another important aspect we need to consider is the keypoint occlusion. We distinguish three types of keypoint occlusion - self-occlusion, occlusion by an object, and keypoints located out of the image bounds. We obviously do not do anything with the out-of-bounds keypoints and they are also not being annotated.

Dealing with the other occlusion types is image dependent. We see that some of the paws in Figure 4 are partially or fully occluded by the snow and are still annotated. Since we have a reference in the other not-occluded paws and some occluded paws are partially visible, we can annotate the points. If the snow was deeper in the image we would not be able to find the tips of the limbs and the keypoints would not be annotated. This also often happens with self-occlusion since the lynxes are usually standing sideways and occluding some part of their body. These occluded keypoints can often be annotated without knowing the precise location of the joint because the lynx's body constitution in a certain pose is predictable. This principle is also applicable to some occlusions by an object. On the other hand, there are circumstances where the occluded keypoint location can not be determined reliably. In that case, the keypoint is annotated as occluded or not annotated at all. Regarding the occlusion during pose estimation itself, we would want the pose estimation model to have low confidence for keypoints that are out of the image or occluded. We hope that the model will be able to have this low confidence score when presented with enough examples that do not contain all keypoints.

3 Deep Learning Methods of Pose Estimation

Nowadays, animal pose estimation problems are most often and most efficiently solved by deep learning methods. As discussed in [29], the Residual Neural Network (ResNet) [24] architecture is commonly used as a backbone to solve this task, supplemented with deconvolution layers as recommended in [70]. a second frequently used and very effective backbone architecture is the High-Resolution Network (HRNet) introduced in [67]. Thus, we now introduce these architectures.

3.1 Residual Neural Network and Simple Baselines for Pose Estimation

Residual networks are well-known architectures introduced in 2015 [24]. This paper addresses the problem of vanishing and exploding gradients when training deep neural networks.

The presented ResNet architecture consists mainly of convolutional layers with 3×3 kernels. The network is formed from multiple convolutional layers operating at the same feature map size followed by downsampling. Additionally, skip connections are used between some layers as we present later. The authors also present two simple rules for building the network regarding the number of filters in layers: “(i) for the same output feature map size, the layers have the same number of filters; and (ii) if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer”. The downsampling between different feature size filters is done by stride 2 convolutional layers.

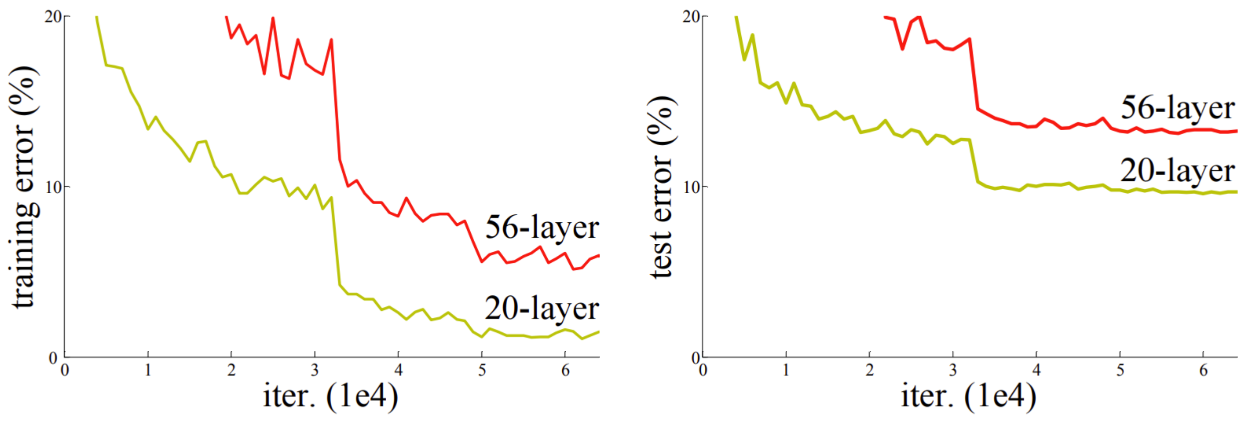


Figure 5: Training and test errors comparison of different depth architectures (20-layer and 56-layer) [24].

In Figure 5, the authors show that for deeper networks, the network’s error rate increases for both training and inference on the test set. However, if a deeper network can be constructed from its shallow counterpart by stacking extra layers, it should at worst have the same error rate. This could be achieved if the stacked layers represented identity mappings in the worst-case scenario. To prevent the phenomenon of the increasing error rate with increased depth, skip connections between the convolutional layers are used to implement said identity mapping.

The authors consider the whole ResNet to be composed of subnetworks that transform the input x by the function $H(x)$. They assume that the neural network is able to approximate residual connections, and instead of letting the network approximate $H(x)$ directly, they re-parametrize the function to its residual form $F = H(x) - x$. Hence the output y of the subnetwork is represented as $y = F(x) + x$. This residual function is implemented in the network using skip connections between the subnetworks. At the same time, the ResNet networks use the ReLU function as a nonlinear activation function. This whole principle is shown in Figure 6 which depicts a basic building block of the ResNet architecture. These building blocks form the subnetworks we mentioned earlier and the network is composed of them. The skip connections are used directly for the same dimensions in the layers as shown in the Figure 6. When the layers change dimension, either the input is padded by zeroes or a linear projection to a higher dimension is used.

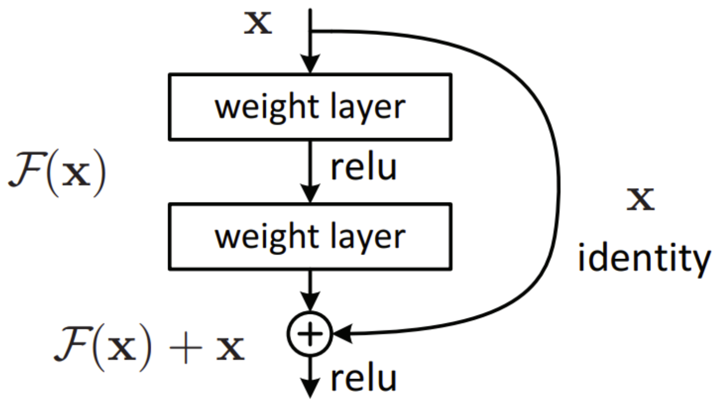


Figure 6: Proposed use of a skip connection in the building block of the ResNet architecture [24].

Five versions of ResNet were proposed in the original paper with 18, 34, 50, 101, and 152 layers, named Resnet-18, Resnet-34, and so on according to the number of layers.

The usage of ResNet networks for pose estimation came a bit later than the original paper. In 2017 it is introduced as a part of the Cascaded Pyramid Network (CPN) [9]. This network has two sub-network parts forming its pyramid network, the first one is called GlobalNet, and the second one RefineNet. ResNet here is used indirectly to build the GlobalNet network.

The main ResNet usage in pose estimation tasks was introduced a year later in the paper “Simple Baselines for Human Pose Estimation and Tracking” [70]. The authors focus on creating simple yet effective baseline methods to battle the increasing complexity of the methods introduced at the time. The pose estimation method is based on the usage of deep and low-resolution feature maps. The deep, low-resolution feature maps are encoded in the last ResNet layers. This method adds three deconvolutional layers behind the last layer with batch normalization and ReLU activation function. The layers have 256 convolutional filters with 4×4 kernel size and stride 2. To generate heatmaps for keypoints a 1×1 convolutional layer is added at the very end of the network with the number of filters equal to the number of the estimated keypoints.

In general, ResNet models have a lower number of Floating-point Operations Per Second (FLOPs) than other networks achieving similar performance. Compared to VGG-19 [60], which is a 19-layer network and has 19.67 GFLOPs, ResNet-152 has 11.58 GFLOPs and better performance on the ImageNet classification task (numbers from MMClassification model zoo [11]). By adding the pose estimation deconvolutional layers from [70], this statement is no longer true. If we compare the two networks presented above - CPN and Simple Baselines ResNet-50 on the COCO validation set in the experiment from [67], we notice that ResNet outperforms CPN with almost a 2% increase in mean average precision score (presented in Section 4.4), but has 8.9 GFLOPs while CPN has only 6.2 GFLOPs.

3.2 High-Resolution Network

The High-Resolution Network is an architecture proposed in 2020 [67]. As opposed to the Simple Baselines principle presented at the end of the previous Section, HRNet attempts to maintain a high-resolution representation of the image during training and inference.

As the authors state, neural networks that use low-resolution representations are the most commonly used type of network for detection, pose estimation, and segmentation tasks. However, this low-resolution provides a coarse output and its performance can be improved by combining it with the middle layers' outputs - medium-resolution representations. Apart from the networks using low-resolution representations, the other commonly used option is hourglass-shaped networks. These networks use a mirrored version of themselves to recover high resolution. An example of such a network is DeconvNet [53] which uses VGG as a backbone and unpooling to recover high resolutions. Another approach is to copy feature maps between layers of the same level and combine this with unpooling. An example of such an architecture is U-Net [57].

There are also architectures that try to work with high-resolution representations - GridNet [17] and multi-scale DenseNet [27]. HRNet attempts to address the weaknesses of these networks. GridNet passes information between low-resolution and high-resolution representations only twice, once from the high-resolution representation to the low-resolution representation and the other way around. The high-resolution representation in the multi-scale DenseNet does not even obtain information from the low-resolution representation and therefore the learned high-resolution representation is not so strong.

HRNet's approach is to maintain a high-resolution representation at all times and gradually add parallel convolutional branches that work with lower-resolution representations. These branches then pass information to each other after a few steps of parallel running. This approach and the HRNet architecture that implements it can be seen in Figure 7.

Before passing the input to the network's main body which is shown in the Figure, the input is reduced to $\frac{1}{4}$ of its original resolution using a stem. Stem consists of two stride-2 3×3 convolutions. The main body then consists of four stages and at the end of each stage, one high-to-low resolution stream is added. The resolution streams operate at $\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}$ resolutions.

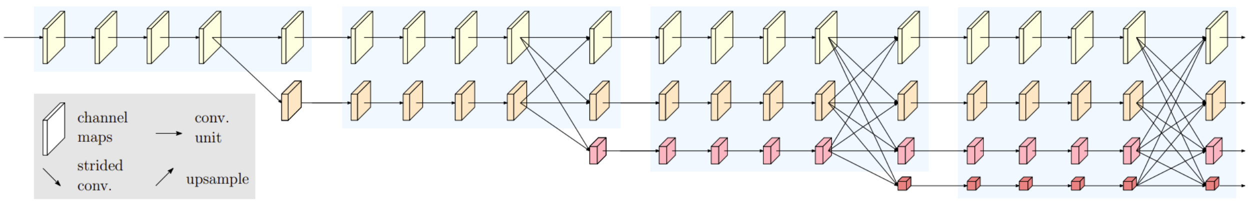


Figure 7: The HRNet backbone architecture with four stages and four parallel branches [67].

As we see in Figure 7, strided convolution and upsampling are used to combine outputs of different resolution streams. The authors call this Repeated Multi-Resolution Fusions and the detail can be seen in Figure 8.

When fusing three representations we have the input set $\{R_r^{\text{in}}\}$, where $r = 1, 2, 3$ denotes the index and also the resolution. It is assumed that $r = 1$ belongs to the first resolution stream and each subsequent stream has a resolution of $\frac{1}{2^{r-1}}$. The output set is $\{R_r^{\text{out}}\}$ and each R_r^{out} is the sum of the transformed inputs as follows:

$$R_{ro}^{\text{out}} = \sum_r f_{r,ro}\left(R_r^{\text{in}}\right), \tag{1}$$

where the function $f_{r,ro}$ depends on the resolution of the input r and on the resolution of the input ro. If $r = ro$ then no transform is needed and $f_{r,ro}\left(R_r^{\text{in}}\right) = R_r^{\text{in}}$. If $r < ro$ then we need to upsample the input and $f_{r,ro}\left(R_r^{\text{in}}\right)$ is bilinear upsampling with 1×1 convolution to match the number of channels. If $r > ro$ then we need to downsample the input and $f_{r,ro}\left(R_r^{\text{in}}\right)$ is $(r - ro)$ times stride-2 3×3 convolution. The fusion formula for two or four representations is derived easily by removing or adding a sum element.

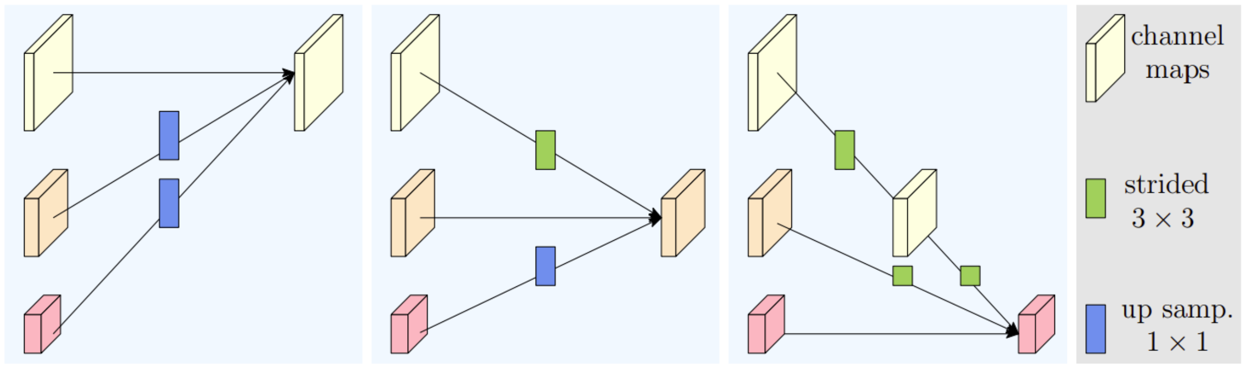


Figure 8: Illustration of the resolution fusion approaches in HRNet for the fusion of different resolution branches. The strided 3×3 means stride-2 convolution of size 3×3, the up samp. 1×1 means bilinear upsampling with 1×1 convolutions to match the number of channels [67].

The authors present three ways to generate the network output - three different representation heads, which are shown in Figure 9. The first output option is the HRNetV1 head, where only the highest resolution representation comes out of the network. The second option is HRNetV2. This head increases the resolution of the low-resolution representations using the previously mentioned bilinear upsampling. It then concatenates the resulting representations and uses 1×1 convolution to mix the concatenated representations. The last head is HRNetV2p, which works in the same way as HRNetV2, but after obtaining the high-resolution representation, it further converts it to low-resolution. This gives a multi-level representation by means of a feature pyramid.

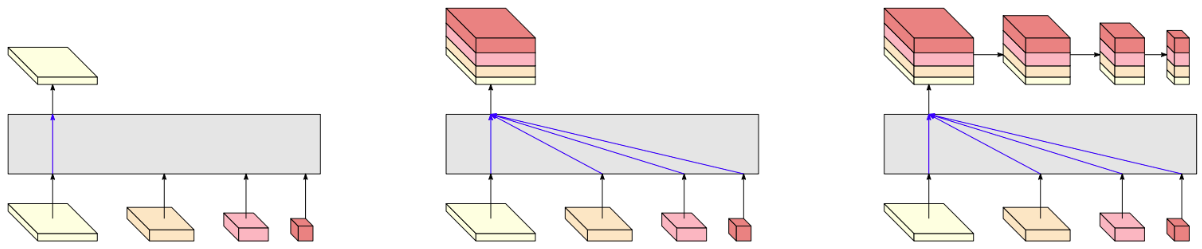


Figure 9: Illustration of the proposed representation heads producing output from the HR-Net network [67]. Left to right: HRNetV1 outputs the highest resolution representation, HRNetV2 outputs concatenated representations (resolutions are upasmpled with bilinear upsampling), HRNetV2p outputs a feature pyramid.

The number of channels for the resolution of the first network stage is given by a parameter C. This width of the convolution is used to calculate the widths for the other stages as $2C, 4C$, and $8C$. The common C values are 18, 32, 48, and 64 and the resulting architectures are then called HRNet-W18, HRNet-W32, and so on.

Having presented the number of GFLOPs in the previous section we can now compare HRNet to the Simple Baselines ResNet according to the experiment from [67]. Again we compare the numbers on the COCO validation dataset. We notice HRNet-W32 outperforms Simple Baselines ResNet-152 with a 2.4% increase in mAP value while having 7.10 GFLOPs, Simple Baselines ResNet-152 has 15.7 GFLOPs. If we also compare the number of parameters we notice HRNet has 28.5 million parameters and Simple Baselines ResNet-152 has 68.5 million parameters, which is almost two and a half times the number of parameters.

3.3 Pose Estimation Heads

Now that we have introduced the neural network backbones, we can look at the various options for applicable heads. There are two approaches in skeletal-based pose prediction, keypoint (coordinates) regression, and heatmap regression.

Deep convolutional neural network (CNN) architectures have started being applied to pose estimation tasks [66] along with the CNNs that started winning image recognition competitions in 2013-2015 [24, 36]. The early networks used the keypoint regression and tried to work with the connection of convolutional layers to fully connected (FC) layers [66], for example by using global average pooling. The keypoint coordinates regression is a product of the last FC layer and the output of the neurons in this layer corresponds to the predicted coordinates in the image. The task of such exact regression is difficult because the network tries to predict real values [3].

The heatmap regression task was first introduced in [65] and it very quickly replaced the coordinate regression usage in the field of pose estimation. The heatmap regression does not utilize global average pooling and FC layers. It instead uses a convolutional layer on the output with channels corresponding to the keypoint heatmaps. This means that estimated joints each have a heatmap that is directly represented by the last layer's channel. The heatmap contains the spatial likelihood of the keypoint location. This means that each heatmap element represents a likelihood of the keypoint being located in the element's position. The regression of the keypoint position $(\hat{x}, \hat{y})$ from the heatmap $\mathbf{H}(x, y)$ is then done as

$$(\hat{x}, \hat{y}) = \underset{x,y}{\operatorname{argmax}} \left(\mathbf{H}\left(x, y\right)\right). \tag{2}$$

As stated in [3], the probabilistic character of heatmaps helps to add extra uncertainty to the keypoint location. Furthermore, heatmaps perform better than networks that directly perform keypoint regression. The disadvantage is the higher number of network parameters.

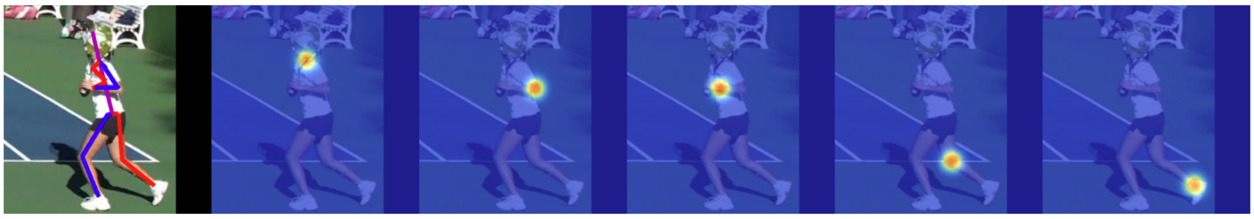


Figure 10: An example of heatmaps output from a CNN. The final pose estimation on the left is obtained as the maximum activation value of the heatmap [52].

Such an example of a heatmap produced by the neural network is shown in Figure 10. We see a few sample heatmaps produced by the Stacked Hourglass Network [52] on the right. On the left, there is a final pose estimation that was regressed from the heatmaps according to Equation 2.

The heatmap generation methods (also called coordinate encoding) and the keypoint deduction from the heatmap (also called coordinate decoding) are also a matter of study [45, 73]. Let us first look at the most basic methods used to encode the coordinates.

The heatmap could be generated as a "one-hot heatmap" (a matrix that has one at the keypoint location and zeroes elsewhere). However, this approach does not help with the aforementioned coordinate regression issues and does not utilize the probabilistic character of heatmaps. Hence, the most basic approach that is widely used is to generate the heatmaps using a Gaussian representation with the mean being in the keypoint coordinates and the variance chosen as a hyperparameter. This approach is sufficient for top-down approaches in which the objects are resized to the same size, but bottom-up approaches need to handle a large object scale variety. This issue is being tackled in [45] by introducing the scale-adaptive and the weight-adaptive heatmap regression. There are also other coordinate encoding shortcomings. For example, the paper [73] tries to tackle the issue of coordinate encoding error introduced by the encoded location quantization.

The coordinate decoding was already introduced in Equation 2. This is the most basic decoding technique which also does not fully utilize the probabilistic character of the heatmaps. The Python framework MMPose [51] uses this decoding as a base for the final keypoint regression. The final pose $(\hat{x}', \hat{y}')$ in the heatmap is regressed as

$$(\hat{x}', \hat{y}') = (\hat{x}, \hat{y}) + 0.25 \left(\operatorname{sign}\left(\Delta_x\right), \operatorname{sign}\left(\Delta_y\right)\right), \tag{3}$$

where sign is the bipolar sign function and Δ_x is the difference calculated from the heatmap $\mathbf{H}\left(x, y\right)$ as $\Delta_x = \mathbf{H}\left(\hat{x}+1, \hat{y}\right) - \mathbf{H}\left(\hat{x}-1, \hat{y}\right)$, the calculation of Δ_y is done similarly.

The paper [52] introduces another basic approach that also uses the second most probable keypoint location and interpolates it with the most probable location. The maximal prediction $\mathbf{m}$ and the second best prediction $\mathbf{s}$ are used to localize coordinates $\mathbf{p}$ as

$$\mathbf{p} = \mathbf{m} + 0.25\frac{\mathbf{s} - \mathbf{m}}{||\mathbf{s} - \mathbf{m}||_2}. \tag{4}$$

These two approaches add sub-pixel precision to the final estimated coordinates and make the predictions more precise. The paper [73] argues that even these decoded locations are coarse. The authors experiment with finding the heatmap extrema by approximating the quadratic form of the predicted Gaussian heatmap by the Taylor series (up to a quadratic term). They do this to formulate the extrema condition (the first derivative being equal to zero) and to find the Gaussian distribution extrema point.

There are several used network heads using the approaches presented above. Possibly the most notable heatmap regression head is the one presented in [70]. We already introduced this heatmap regression approach in Section 3.1 where the heatmaps are generated by using k channels of 1×1 convolutional networks for k keypoints. HRNet [67] uses a very similar approach and regresses the heatmaps from the high-resolution output. Almost the same approach is presented in Stacked Hourglass Network [52] which uses two consecutive 1×1 convolutions to produce the predictions.

The keypoint regression heads are a matter of study too because the heatmap approach also has major drawbacks. As [61] states, probably the biggest issue that comes with heatmaps is the non-differentiability of the argmax operation. This leads to the construction of loss functions to optimize the heatmaps and not the final coordinates. Another problem is the already mentioned quantization error due to the lower resolution of the produced heatmaps. This could be fixed by increasing the heatmap resolution, but the increased resolution brings computational difficulties and a quadratic increase in model complexity. This model complexity is also the reason why keypoint regression methods are widely used in 3D pose estimation [61].

The first regression head we look at consists of a simple pooling layer followed by three FC layers, the first and the second FC layer has 4096 neurons, and the last layer has $2k$ neurons for coordinate regression. This regression head was introduced in [66]. The authors of [61] try to find a connection between keypoints and heatmaps and estimate the joints by integrating the heatmap locations weighted by their probability. The heatmap is normalized by soft-max operation to make its elements non-negative and sum to one. This is called the Integral Pose Regression and can be applied to any heatmap-based method. Another approach introduced in [39] reformulates the regression problem as a maximum likelihood estimation task and tries to regress the underlying ground truth distribution. They come up with a Residual Log-likelihood Estimation which helps regress both the coordinates and their variance. The head is implemented as average pooling followed by an FC layer which consists of $k \times 4$ neurons, two neurons are for the coordinates and the remaining two for mean and variance of each keypoint.

Loss functions

Let us now briefly introduce the common loss functions used to train the pose estimation models. First, we introduce the mean squared error (MSE) loss for heatmap regression, and then we introduce the L_1 and smooth L_1 losses for keypoint regression.

The MSE loss is used to minimize the mean squared error between each prediction and target. The squared error between the prediction x_i and the target y_i is calculated as

$$\mathrm{SE}\left(x_i, y_i\right) = \left(x_i - y_i\right)^2. \tag{5}$$

If the target is the heatmap of the size $m \times n$, then the operation is done element-wise and the result is also of the size $m \times n$. The output is then reduced either by taking the sum of all elements or the mean of all elements. In popular neural network implementation frameworks like PyTorch, the loss is called MSE even when only the elements' sum is taken. The final loss for K heatmaps is then usually a weighted sum of the partial losses as

$$\mathrm{MSE}\left(x, y\right) = \sum_{k=1}^{K} w_k \frac{\sum_{m_k} \sum_{n_k} \left(x_{m_k,n_k} - y_{m_k,n_k}\right)^2}{mn}, \tag{6}$$

where x_{m_k,n_k} is the predicted value on the position (m, n) in the predicted heatmap and y_{m_k,n_k} is the target value on the position (m, n) in the target heatmap for keypoint k. The w_k is the weight of each partial loss and is often set to be equal to one.

The L_1 loss is used to minimize the absolute error between the prediction and the target. The absolute error between the prediction x_i and the target y_i is calculated as

$$\mathrm{AE}\left(x_i, y_i\right) = \left|x_i - y_i\right|, \tag{7}$$

and if the targets are keypoint coordinates, then the absolute error is calculated for each coordinate element. As with the MSE loss, the L_1 in the PyTorch framework is then calculated either as the sum of all the elements or their mean. The smoothed version of the L_1 loss is calculated as

$$l_i(x_i, y_i) = \begin{cases} \frac{0.5(x_i - y_i)^2}{\beta} & \text{if } |x_i - y_i| < \beta \\ |x_i - y_i| - 0.5\beta & \text{otherwise} \end{cases}, \tag{8}$$

where β is usually chosen as $\beta = 1$. The mean of all coordinates is usually taken for the final loss and the loss is then calculated as

$$\text{smooth-}\mathrm{L}_1(x, y) = \frac{\sum_{i=1}^{N} l_i(x_i, y_i)}{N}. \tag{9}$$

In the Equation we see that the smoothed version behaves as the L_2 loss when the absolute error term is lower than β and behaves as the L_1 loss otherwise. This makes the function differentiable and also makes it more robust to outliers (because for big differences $x_n - y_n$, the error is not being squared).

4 Evaluation

To properly evaluate the pose estimation models' performance, we now present the methods used to evaluate pose estimation.

Two main approaches are used to evaluate pose estimation. The first approach is to evaluate each keypoint separately, which is for example done by the percentage of correct keypoints (PCK) metric. This gives us a number indicating the performance of the model for each keypoint. The second approach is to use the evaluation for each keypoint and calculate the overall model performance.

In the following sections, we first discuss the first approach and introduce the frequently used PCK metric. We then modify this metric as a percentage of correct predictions and use it to evaluate the overall model success rate. The percentage of correct predictions already falls under the second approach mentioned above. Finally, we present the well-known OKS metric and its different interpretations with their advantages and disadvantages. The OKS metric is used to calculate mean average precision for pose estimation tasks.

4.1 Percentage of Correct Keypoints

The percentage of correct keypoints metric is a widely used evaluation metric for pose estimation tasks. As mentioned, this metric measures the accuracy of individual keypoint predictions. It is defined as the percentage of predicted keypoints whose distance from ground truth is less than a given threshold distance t. We need to keep in mind that this percentage is measured separately for each keypoint.

Thus, for each detected keypoint, we compute the PCK as the ratio of the number of predictions satisfying this condition to the total number of predictions. The calculation for keypoint i is as follows:

$$PCK\left(i,t\right)=\frac{\sum_{i=1}^{N}\delta\left(d_i,t\right)}{N}, \tag{10}$$

where d_i is the Euclidean distance between i-th ground truth keypoint and the i-th predicted keypoint, N is the total number of predictions, and $\delta\left(d_i,t\right)$ is a function defined as follows:

$$\delta\left(d_i,t\right)=\begin{cases}1 & \text{if } d_i<t\\ 0 & \text{otherwise}\end{cases}. \tag{11}$$

4.2 Percentage of Correct Predictions

We can now use the already defined PCK to evaluate the overall success of pose estimation. To do this, we use the condition defined in the PCK. We check that all predicted keypoints within one pose prediction are closer to the ground truth than a given distance threshold. By checking this condition, we obtain information about the pose predictions correctness at the given threshold t. If we calculate the ratio of these correct predictions to the total number of predictions, we get the percentage of correct predictions at the threshold. We obtain it using the modified formula as follows:

$$P\left(t\right)=\frac{\sum_{i=1}^{N}\operatorname{sign}^{+}\left(\prod_{k=1}^{Nk}\delta\left(d_{i,k},t\right)\right)}{N}, \tag{12}$$

where sign$^+$ is the unipolar sign function, $d_{i,k}$ is the Euclidean distance between k-th ground truth keypoint and the k-th predicted keypoint for the i-th ground truth image. We notice that the product iterates over all Nk predicted keypoints within one ground truth image. The sum then iterates over all N images. Again we use the $\delta\left(d_{i,k}, t\right)$ function presented in Equation 11.

4.3 Object Keypoint Similarity

Object keypoint similarity (OKS) is an evaluation metric that, like other similarity metrics, gives us a number in the interval $\langle 0, 1\rangle$ indicating the estimated pose similarity to the ground truth data. The perfect prediction has a similarity score of 1. OKS is presented by the authors of the COCO dataset [41] as the main metric to evaluate pose estimation.

The OKS authors say that its goal is to perform the evaluation in a similar way to the object detection evaluation, i.e., to allow for average precision (AP) and average recall (AR) calculation. To evaluate AP and AR a similarity measure needs to be used. In the case of detection evaluation, the similarity measure is Intersection over Union (IoU). With IoU the curves for precision and recall can be computed by choosing different IoU thresholds. Thus OKS is defined as a similarity measure for keypoint detection.

OKS for one vector of keypoint predictions $\mathbf{p}$ and ground truth keypoints $\mathbf{g}$ for image $\mathbf{I}$ is calculated as follows:

$$OKS\left(\mathbf{p}, \mathbf{g}\right) = \frac{\sum_i \exp\left(\frac{-d_i^2}{2s^2\kappa_i^2}\right)\delta\left(0, v_i\right)}{\sum_i \delta\left(0, v_i\right)}, \tag{13}$$

where d_i is the Euclidean distance between the corresponding i-th predicted keypoint and ground truth from vectors $\mathbf{p}$ and $\mathbf{g}$, s is the ground truth object scale, κ_i is the per-keypoint constant. The ground truth keypoint visibility flag v_i is greater than 0 if the keypoint is visible and equal to 0 if not. We assume that at least one keypoint is visible to avoid division by zero and to make the pose estimation task meaningful.

The κ_i value represents the keypoint's i unceraínty. It is calculated using a set of redundantly annotated images, meaning multiple annotators have to annotate the same image set. From these redundant annotations, we compute the variance σ^2 of the annotated keypoints positions with respect to the scale of the object. We then determine κ_i for each keypoint as $\kappa_i = 2\sigma$. This means that κ_i values depend on the object type for which we are estimating the pose. This dependency and the cost of annotating images redundantly makes these values difficult to obtain and they are often approximated from existing values computed for some benchmark datasets.

The object scale s can also be defined in different ways. In the official COCO definition, it is the square root of the object segment area. If one does not have segmentation for the object, it is the square root of the bounding box area. However, the object size introduced this way does not take into account different image sizes and box-to-image ratios. Thus, one can also encounter versions of OKS where the object scale s is introduced as the ratio of the bounding box size to the total image size.

However, defining the scale s with respect to the image size introduces problems with numerical stability. Thus, we propose two approaches to account for image size. The first is to use a correction factor to solve the numerical problems. The second is to use the standard definition with the addition of a constant representing different image sizes.

4.3.1 Scale-aware OKS

As mentioned earlier, we propose two solutions to account for image size in the OKS calculation.

The first solution is to modify the meaning of the scale term s in the OKS calculation formula. Instead of the square root of the object segment area, we use the square root of the ratio of the bounding box size and the image size. However, using this solution easily leads to numerical instability.

In the OKS calculation, we use an exponential whose exponent is always negative because the Euclidean distance d, the scale s, and the per-keypoint constant κ are always positive numbers. At the same time, the scale s is always a number lower than one. The same statement applies to the per-keypoint κ constant. Moreover, both of these numbers are squared in the denominator. Thus, there is a large negative number in the exponent. When the bounding box is small compared to the image size, the result of the exponential term gets so close to zero that the numerical precision of floating-point numbers is not sufficient enough to account for it and the result gets rounded to zero.

We address this numerical instability by adding a correction factor c to the OKS formula. This correction factor must be constant across the evaluated problem, otherwise, the OKS results would not be mutually comparable. It is then added to the formula as follows:

$$OKS\left(\mathbf{p},\mathbf{g}\right)=\frac{\sum_i \exp\left(\frac{-d_i^2}{2cs^2\kappa_i^2}\right)\delta\left(0,v_i\right)}{\sum_i \delta\left(0,v_i\right)}. \tag{14}$$

As mentioned above, it is essential that the correction factor c is fixed for the results to be comparable. If we then wanted to interpret the results across experiments that use OKS, the correction factor value must of course be the same.

The second solution is to use the scale s calculation proposed by the OKS authors. The scale s is therefore still the square root of the size of the object segment area or the square root of the size of the bounding box. We then modify the formula by adding the image ratio term r. We the calculate OKS as follows:

$$OKS\left(\mathbf{p},\mathbf{g}\right)=\frac{\sum_i \exp\left(\frac{-d_i^2}{2crs^2\kappa_i^2}\right)\delta\left(0,v_i\right)}{\sum_i \delta\left(0,v_i\right)}. \tag{15}$$

We kept the correction factor c in the formula to compare it to the calculation from Equation 14.

We can calculate OKS at different correction factor values and show why it is needed only in the first proposed formula. The image ratio term r also needs to be calculated for every image. It gets calculated as follows:

$$r=\frac{I_r}{I_c}, \tag{16}$$

where I_r is the reference image size and I_c is the size of the image the current prediction is made for. We propose to choose the reference image size as the median value of the image sizes in the given image set.

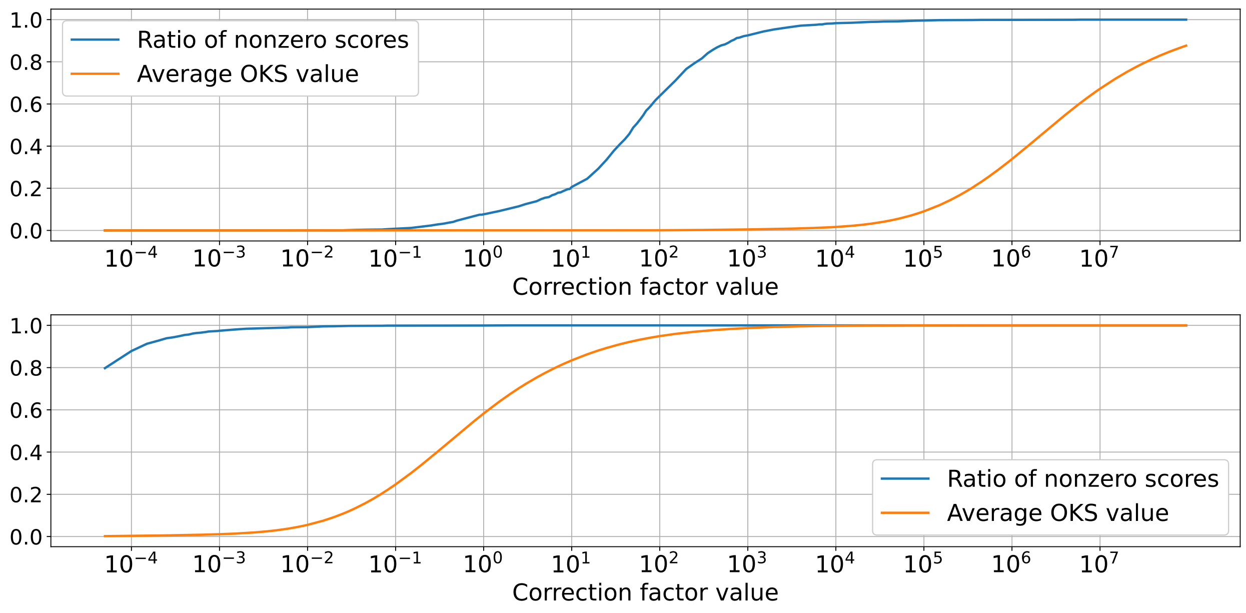


Figure 11: The experiment showing the correction factor effect. The top graph OKS values are calculated as in Equation 14, and the bottom graph OKS values are calculated as in Equation 15. The "Ratio of nonzero scores" is the relative amount of images with OKS score greater than zero. The "Average OKS value" is the average of the values for the images in the set.

We depict the experiments with scale-aware OKS implementations in Figure 11. These experiments are done on our Lynx-Pose dataset images presented later in Section 5.1. The model we use to make the predictions is HRNet-W32 with Simple Baselines heatmap regression head pre-trained on the Animal-Pose dataset presented in Section 5.2. The top graph in the Figure 11 is the scale-aware OKS using the bounding box to image size ratio presented in Equation 14. The bottom graph is the scale-aware OKS using the square root of the bounding box area with the image ratio term presented in Equation 15. As the I_r value we use the median image size in the dataset. As we see in the bottom graph the correction factor is not really needed for the image ratio implementation. The ratio of nonzero scores to the total number of images is nearing 1 and the average OKS value is greater than 0.5. On the other hand, we notice that the top graph has a ratio of nonzero OKS scores lower than 0.1. This means that more than 90% of the calculated values got floored to zero due to numerical instability for the correction factor equal to one. We achieve above 60% of nonzero values if we increase the correction factor to 10^2. If we keep increasing the factor we can analyze the graph to find a good correction factor value for the given experiment. This value corresponds to a factor where the ratio of nonzero scores is nearing one and the average OKS value is also acceptable - this correction factor value would be around 10^5 for our experiment.

4.4 Average Precision for Pose Estimation

As mentioned at the beginning of Section 4.3, the OKS value is defined mainly for us to be able to calculate the standard performance metrics - Average Precision and Average Recall. In fact, the Average Precision (AP) notation used for pose estimation stands for mean Average Precision (mAP), and as COCO authors state [41], these terms are often used interchangeably and depend on context.

To calculate the mAP value, we need to first calculate Average Precision at a given threshold T as follows:

$$AP(T) = \frac{\sum_{i=1}^{N} \delta\left(OKS\left(p_i, g_i\right), T\right)}{N}, \tag{17}$$

where N is the total number of predicted poses. This gives the relative count of predictions that had an OKS value greater than the threshold T.

The mAP is then the mean of AP values at different thresholds as follows:

$$\text{mAP}\left(\{\mathbf{T}\}\right) = \frac{\sum_{T \in \{\mathbf{T}\}} AP\left(T\right)}{|\mathbf{T}|}, \tag{18}$$

where $\{\mathbf{T}\}$ is the thresholds set for which we calculate AP values and $|\mathbf{T}|$ is the set's magnitude. The thresholds set for the commonly used COCO evaluation [41] is given as $T = \{0.5, 0.55, 0.6, \ldots 0.95\}$, this set used in mAP calculation is often denoted as AP at OKS=.50:.05:.95. It is common to evaluate only AP with single threshold values of $T = 0.5$ and $T = 0.75$ too. Another usual evaluation of mAP takes the object size (ground truth bounding box size) into account and evaluates mAP at different object scales. Small objects have an area lower than 32^2 pixels, medium objects have an area between 32^2 and 96^2 pixels, and large objects have an area greater than 96^2 pixels.

5 Datasets

Several datasets are available for animal pose estimation problems. These datasets focus on pose estimation tasks of multiple animals and contain various animal species. In our work, we want to focus on datasets containing mammals. We look for a dataset containing a quadruped mammal species that have their pose defined by the same keypoints we mention in Section 2.1. Another aspect we need to consider is the given species' similarity to lynxes. These aspects allow us to choose a dataset and to look for models pre-trained on a given dataset. We hope these mentioned aspects would make the finetuning better and more robust.

In this section, we first introduce our novel dataset for lynx pose estimation and its characteristics, then we look at the other available datasets regarding the animal pose estimation task.

5.1 Lynx-Pose dataset

The Lynx-Pose dataset is a novel dataset we present containing the images of the Eurasian lynx photographed by camera traps in the national parks Šumava and Bayerwald. The images capture these rare animals in the wild and in various natural conditions.

The dataset contains 2,151 annotated images of various sizes depending on the camera trap that took the image. The image sizes range from 640×480 pixels to $5,152 \times 3,968$ pixels and have another 16 unique resolutions between these two sizes. This means that the dataset allows for tackling the problem of multi-scale object detection, segmentation, and pose estimation.

The bounding boxes and segmentations are annotated for all lynx images in the dataset. In the 2,151 images, there is a total of 2,208 annotated lynxes, we notice that very few images contain multiple lynxes as per the assumption made in Section 2.1.

The lynx's distance from the camera also varies greatly throughout the dataset which, combined with the varying image sizes, results in a bounding box sizes variability. The different sizes of the lynx individuals also contribute to this variability. The bounding box area size is usually used to determine whether the object size is small, medium, or large as stated in [41] and in Section 4.4. The small objects have an area lower than 32^2, medium objects have an area size between 32^2 and 96^2, and large objects are the objects with an area greater than 96^2 pixels. All of the lynxes in our dataset are large objects according to this measurement metric making this size system useless for our issue.

Hence we plot the bounding box areas in Figure 12. The histogram shows counts of bounding boxes with size within a given interval relative to the biggest bounding box size. We observe a big variance in the box sizes. The biggest bounding box in our dataset has an area of $19,548,447$ pixels. The smallest bounding box has an area of $4,614$ pixels. We also notice that most of the boxes lie in the 0% to 20% area. We show the closeup of this area in the same Figure. With the information available, we could now redefine the small objects for our task as the ones having an area less than 2.5% of the biggest bounding box area, the medium objects could lie between 2.5% and 20% of the biggest bounding box area and the rest could be large objects. This way, the object sizes classification can be made to fit the task at hand and makes more sense than the rigid object size classification proposed in [41].

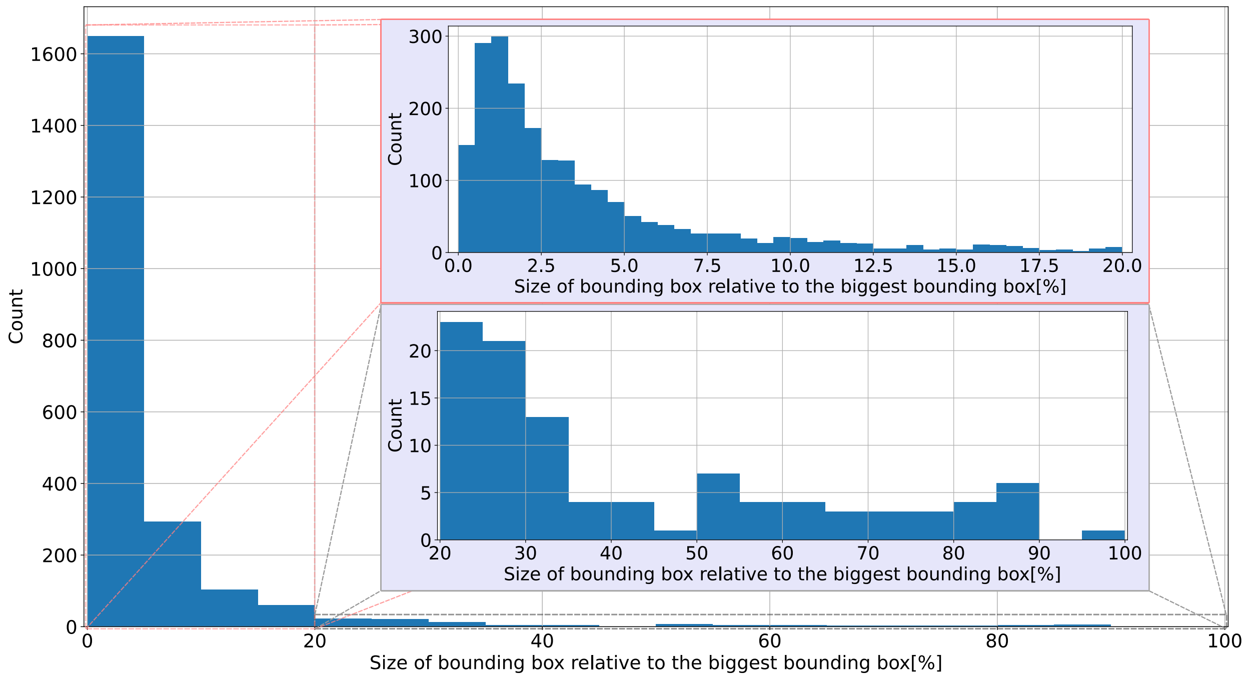


Figure 12: Histogram of the bounding box size distribution within the animal pose dataset relative to the size of the biggest bounding box. More detailed depictions of ranges 0% to 20% and 20% to 100% are shown in the closeups

Out of the 2,208 annotations, not all include the annotated pose. Sometimes, the lynx is blurred because of its movement and the pose is impossible to annotate. The other issue is that sometimes the lynxes are stood in a way that the pose can not be annotated either. Such lynxes have the "NOT_FOR_POSE" tag in their annotation meaning that the pose is not annotated for the given image. Some of the images are missing the "NOT_FOR_POSE" tag but also do not have the pose annotated at all. a total of 1,689 lynxes are annotated for pose estimation in 1,685 images.

The pose itself is given by 20 keypoints mentioned in Section 2.1. These keypoints are two ears, two eyes, a nose, withers, a neck, a tail base, four paws, four "knees", two shoulder points, and two hip points. As we see in Figure 13, most of the annotations (69.2%) have all 20 keypoints annotated and the majority of the annotations (90.2%) have pose at least 18 keypoints annotated.

Another benefit of our dataset is the lynxs' identity annotation. When the lynx as an individual is recognizable in the image, its identity is annotated as well. There is a total of 18 identified lynx individuals across the dataset in a total of 1,386 images. This allows for the use of the dataset to train a re-identification model which is needed to track lynxes successfully as was mentioned in the Introduction.

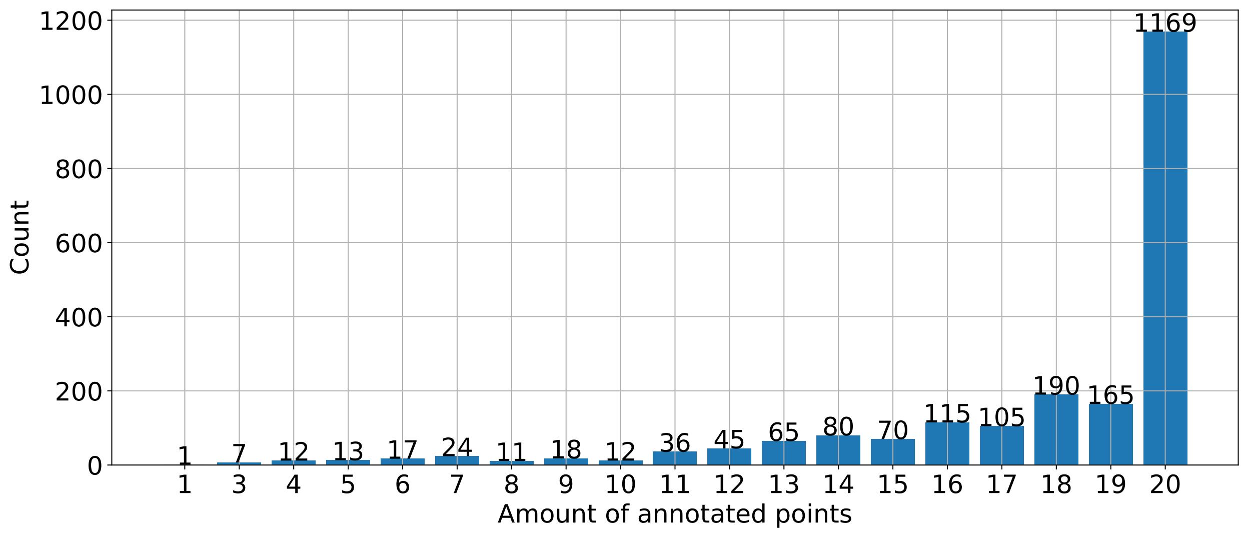


Figure 13: Absolute count of the annotations containing a given amount of annotated keypoints.

As we already mentioned the dataset contains lynxes in various conditions. This can be seen in Figures 14 and 15. We see that the dataset captures the lynxes during the daytime, evening, and night. Another important aspect is the time of the year because lynxes mostly live in temperate climate zones. Our dataset captures the temperate weather, in the Figures we see images with and without snow, with and without greenery, and with and without sunshine. These aspects are of great importance for image classification as shown in [56] on the famous example of the "Husky vs Wolf" experiment. The classification task is of great importance for determining if the image should be passed to an object detector model and subsequently to a pose estimation model.

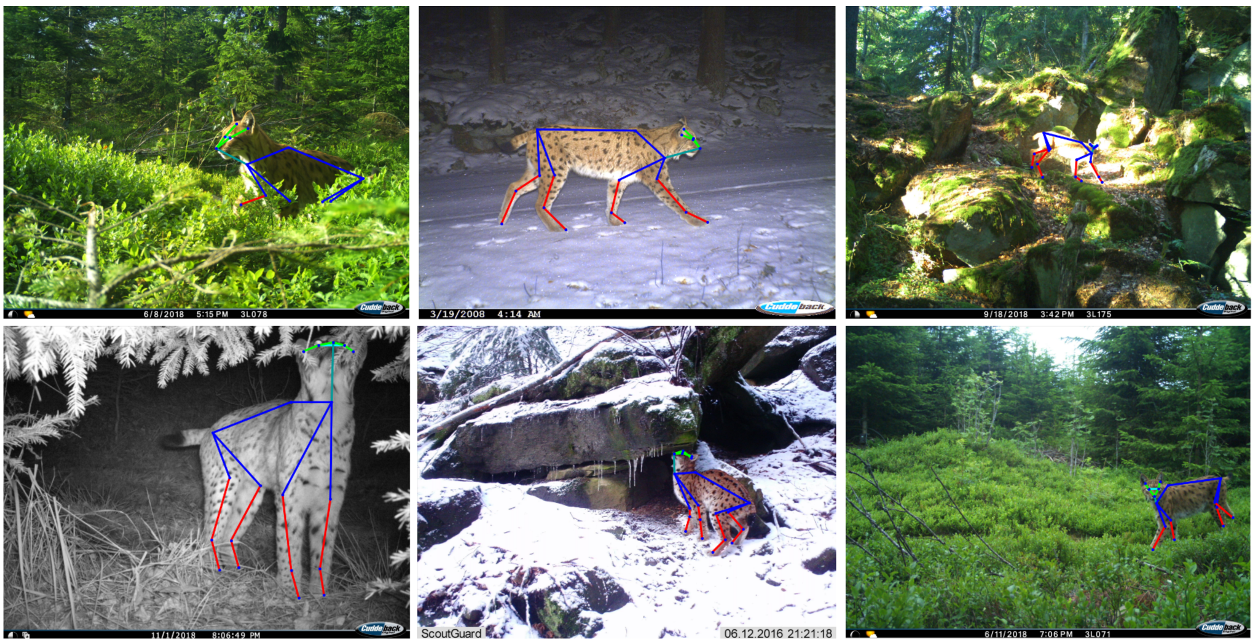


Figure 14: Example of annotated pose keypoints in Lynx-Pose dataset.

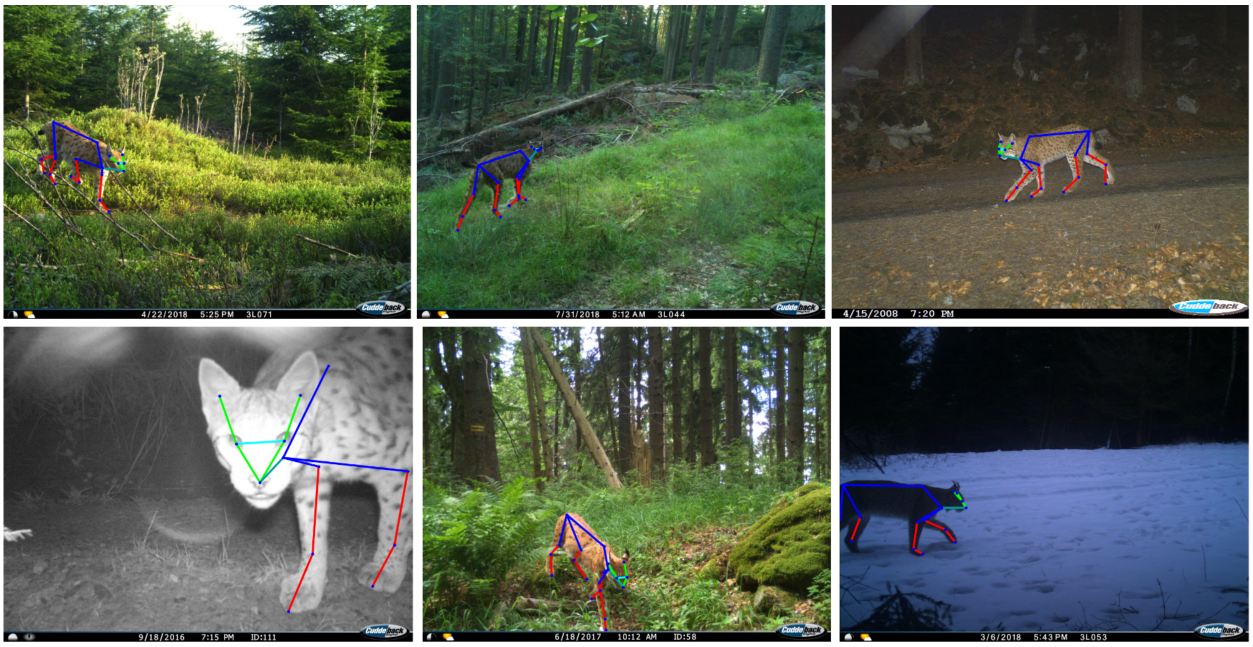


Figure 15: Example of annotated pose keypoints in Lynx-Pose dataset.

5.2 Animal-Pose dataset

The Animal-Pose dataset was proposed in paper “Cross-Domain Adaptation for Animal Pose Estimation” in 2019 [6]. It contains images from the publicly available PASCAL Visual Object Classes Challenge 2011 dataset [15]. Five mammal categories were selected for the dataset and their pose was annotated. The chosen animals are the following: cat, dog, horse, sheep, and cow.

This dataset contains 4,608 images of the previously mentioned animals with 5,517 animal instances throughout the images. The size of these images varies, but the image size is usually around 500 × 500 pixels. As the authors state, since animal pose datasets are scarce, they use a novel cross-domain adaptation method to transform the pose knowledge from labeled animal species to unlabeled animal species. This is done with the proposed dataset combined with a human pose dataset from which the pose knowledge is also transferred.

The pose is represented by 20 keypoints, the final number of keypoints in the images can be smaller because of occlusion or out-of-bounds keypoints. The annotated keypoints are as follows: four paws, two eyes, two ears, four elbows, nose, throat, withers, tail base, and the four knee points. a keypoint annotations example from this dataset can be seen in Figure 16.

If we compare the defined keypoints with our Lynx-Pose dataset keypoints from Section 5.1, we notice the similarity between the datasets. We can also compare Figures 14 and 15 with Figure 16 and notice that the pose looks almost the same when visualized. Since the dataset contains a cat category and quadruped mammals in general, it is a good candidate dataset for which we can find pre-trained models. However, there are subtle differences that will introduce an initial error. The first one is the paws annotation. We notice the cats’ and the dogs’ paws not being annotated at the tip of the limb in the Animal-Pose dataset. The other, less meaningful difference is the animals’ skulls. We notice a difference in the position of the ears related to the position of the eyes for horses and sheep. This difference should not matter that much since the dataset contains cats, which are similar to the lynxes and hopefully generate similar features in the neural network feature maps.

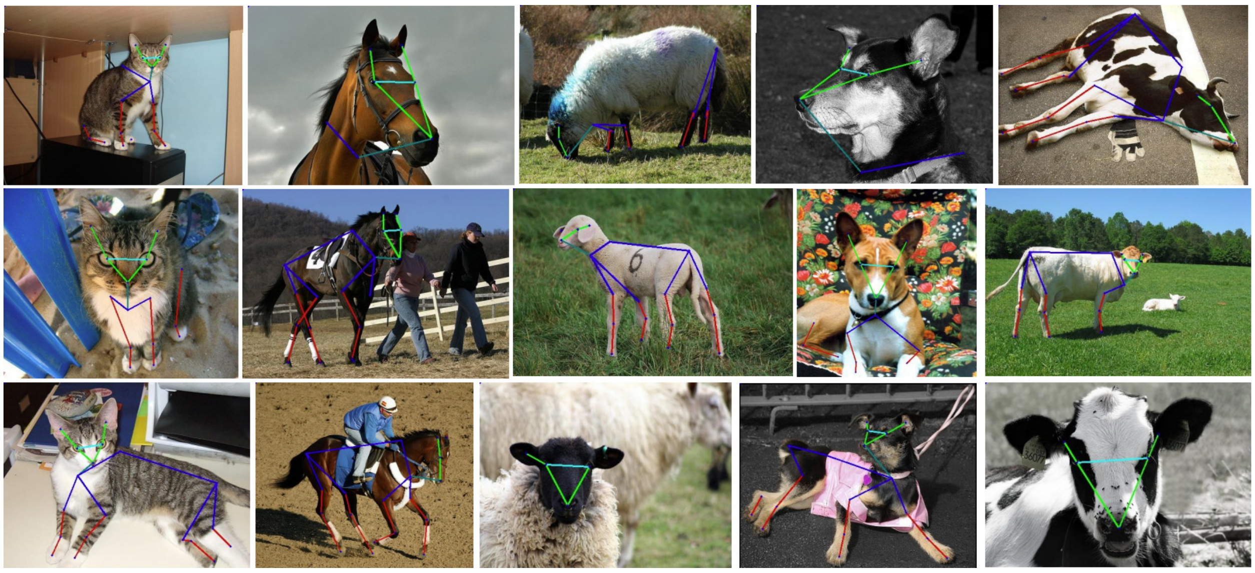


Figure 16: Example of annotated pose keypoints in Animal-Pose dataset [6].

5.3 Amur Tiger Re-identification in the Wild dataset

Amur Tiger Re-identification in the Wild dataset (ATRW dataset) was introduced in 2019 in “ATRW: a Benchmark for Amur Tiger Re-identification in the Wild” [40]. It contains Amur tiger images from large zoos because these tigers are on the brink of extinction in the wild.

The dataset contains 8,076 video clips containing at least one Amur tiger uniformly sampled into frames. These images are of varying resolutions. The resulting dataset with pose keypoints annotated contains 4,126 images. There is also a re-identification dataset created in which the images are cropped to only contain the tiger.

Multiple tiger attributes are annotated in this dataset. For each tiger, a bounding box and the tiger’s view orientation (frontal, left, right, and back) are annotated. Tigers also have their identity annotated or have an unknown id annotation if the identity cannot be determined. The orientation of the tiger is very important because the tiger’s stripe patterns are different on each side. This results in one tiger being treated as two different entities in the re-identification dataset when viewed from two different sides. The entities are then linked to one tiger identity in the dataset.

Finally, tigers have their pose keypoints annotated and the following 15 keypoints are used: two ears, a nose, two shoulders, two hips, four paws, two back knees, a tail base, and a midpoint between the nose and tail base. These annotations can be seen in Figure 17.

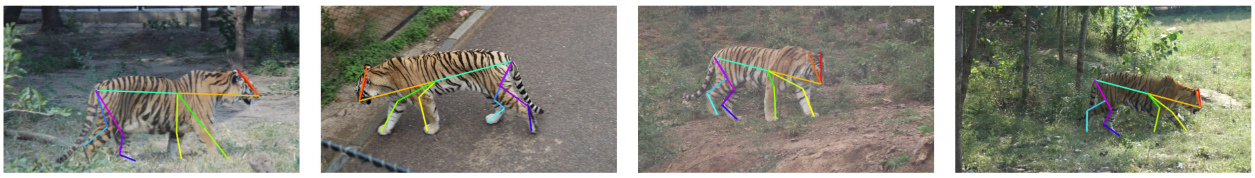


Figure 17: Example of annotated pose keypoints in ATRW dataset [40].

5.4 AP-10K dataset

AP-10K dataset was proposed in August 2021 as a benchmark dataset for general animal pose estimation. It was proposed in the paper “AP-10K: a Benchmark for Animal Pose Estimation in the Wild” [72]. It aims at the benchmark dataset construction which would allow the models to have better generalization ability on unseen animal species.

AP-10K dataset contains around 60,000 images, 10,015 images are annotated with pose annotations, and around 50,000 images are organized into animal families without keypoint annotations. There are 54 animal species from 23 animal families. We notice a huge increase in the number of animal species in comparison to the previously introduced datasets. This diversity is introduced to answer the questions of whether or not the pose estimation model benefits from a large-scale multi-species dataset and how much impact pre-training on human pose estimation datasets has on the performance.

Pose of the animals is represented by the following 17 keypoints: two eyes, nose, neck, tail, two shoulders, two elbows, two knees, two hips, and four paws. These annotations are shown in Figure 18.

Background types are also labeled for all images containing pose annotation. These background types are grass or savanna, forest or shrub, mud or rock, snowfield, zoo or human habitation, swamp or riverside, desert or gobi, and mugshot.

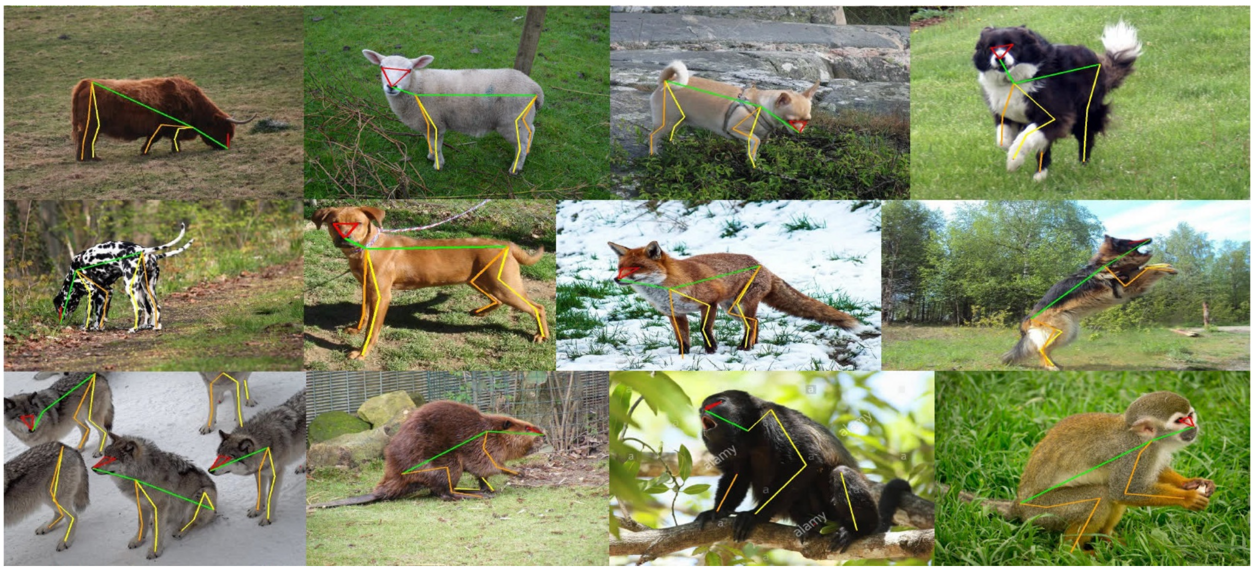


Figure 18: Example of annotated pose keypoints in AP-10K dataset [72].

5.5 Horse-10 dataset

Horse-10 dataset is from the year 2021 and was proposed in the paper “Pretraining Boosts Out-of-Domain Robustness for Pose Estimation” [48]. The paper explores the question of how well an algorithm that performs with high accuracy on an individual animal can generalize its estimation to different individuals with different appearances. Because of this, the dataset contains only 30 diverse thoroughbred horses. The horses have various appearances (coat colors) and are captured at different farms to add location complexity. The dataset was split into 3 parts that contain 10 randomly selected training horse individuals (hence called Horse-10). The other two splits were used for evaluation on out-of-domain horses.

The dataset consists of 8,114 images of these horses. Because of the scarce amount of individuals, the pose estimation algorithm needs to be trained on a lot of possible movement patterns. The dataset images are taken from videos capturing moving horses and their size is 288 × 162 pixels.

Horse pose in this dataset is represented by 22 keypoints. The images in the dataset are also "corrupted" by 15 forms of digital transformations, blurring filters, point-wise noise, or simulated weather conditions. This way the Horse-C dataset is created. The authors created the Horse-C datasets using each corruption at 5 different severity settings resulting in a total number of 75 datasets and over 600,000 images. The example of the corrupted images from the dataset are shown in Figure 19.

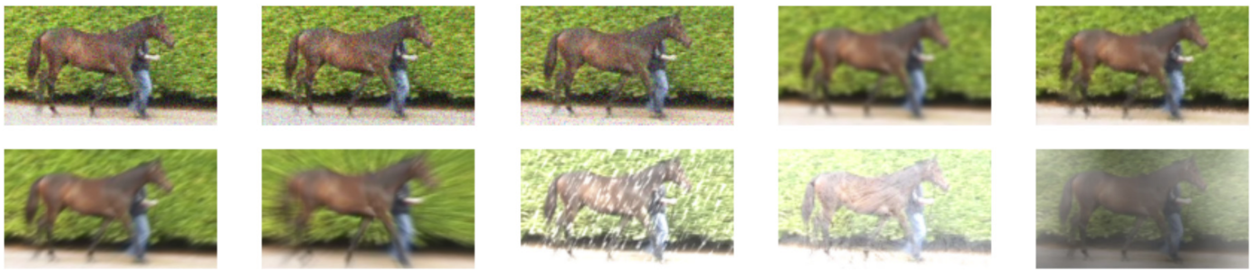


Figure 19: Example of corrupted images from the Horse-C dataset [48].

5.6 MacaquePose dataset

The MacaquePose dataset is a novel dataset created and described in "MacaquePose: a Novel "In the Wild" Macaque Monkey Pose Dataset for Markerless Motion Capture" [37]. This dataset contains macaque images from the internet, images captured in zoos, and images captured at the Primate Research Institute of Kyoto University.

It contains 13,083 images with 10,630 images containing a single monkey, and 2,453 pictures containing multiple monkeys.

The pose of the macaques is represented by 17 keypoints: nose, two ears, two eyes, two shoulders, two elbows, two wrists, two hips, two knees, and two ankles. An example of keypoint annotations from this dataset can be seen in Figure 20.

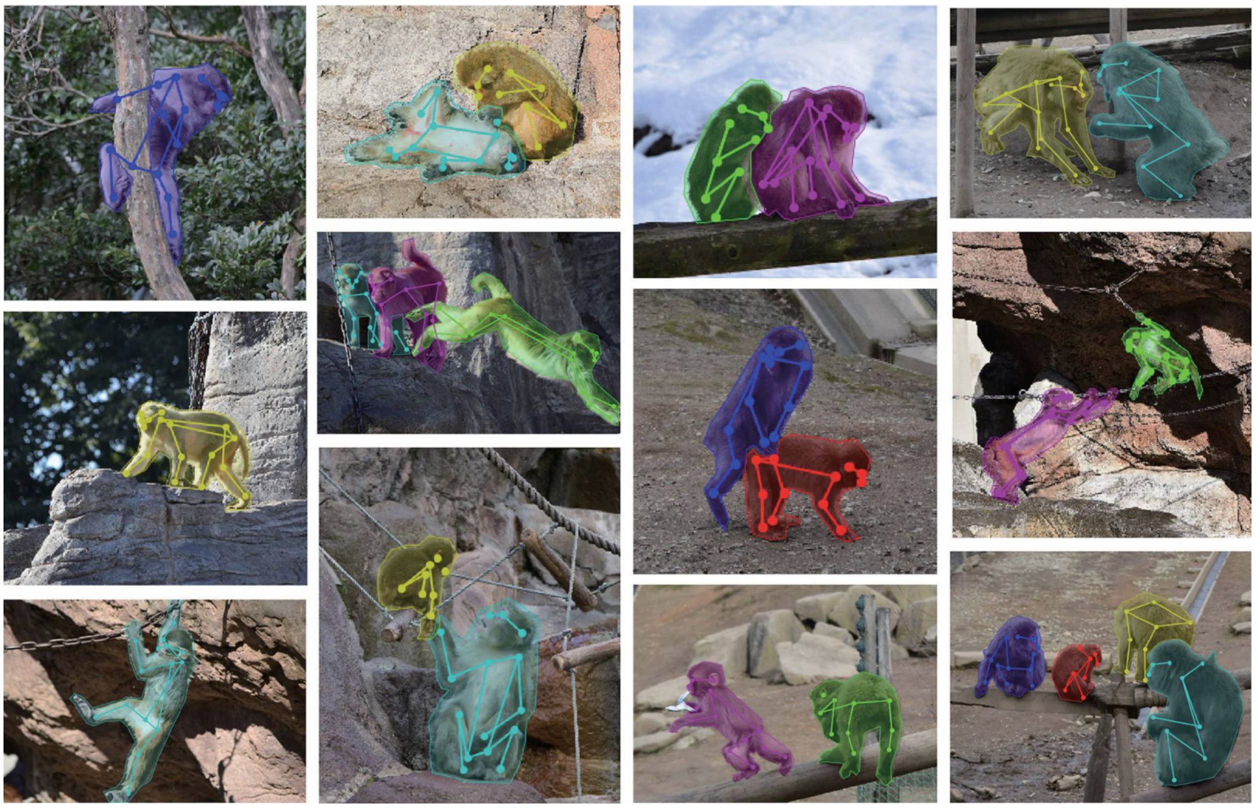


Figure 20: Example of annotated pose keypoints in MacaquePose dataset [37].

5.7 Other datasets

There are also other datasets regarding animal pose estimation problems that are not as interesting for the task of lynx pose estimation.

a paper “Fast Animal Pose Estimation Using Deep Neural Networks” [54] published in 2018 introduced the LEAP method for animal pose estimation for behavioral analysis. It focused on fruit fly pose estimation and fast prediction on new data.

In October 2019 the paper “DeepPoseKit, a Software Toolkit For Fast And Robust Animal Pose Estimation Using Deep Learning” [22] proposed a deep learning toolkit for tracking animals’ movements. This toolkit was tested on the pose and movement estimation of fruit flies, locus, and Grévy’s zebras when viewed from above.

Another dataset regarding behavioral analysis of insects was proposed in “DeepBees – Building and Scaling Convolutional Neuronal Nets For Fast and Large-scale Visual Monitoring of Bee Hives” [47] and focuses on bee pose estimation.

6 Image augmentations

Image augmentations are a way to generate new training data with the use of various image preprocessing techniques such as brightness transformations, geometric transformations, convolutions with various convolutional kernels, or other more complicated operations. As mentioned in the Introduction, image augmentations (or image transformations) can help greatly with the performance the model is able to achieve. Since augmentations bring an aspect of randomness into training and generate "new" data, they are a great tool for improving the ability to generalize [50, 63, 76]. The images can be augmented prior to the training hence the generated training data are always the same. The second option is to use online generation during the training and to have differently augmented data for each epoch.

There are a lot of image-processing libraries implementing the augmentations. In our work, we focus on two implementations and the image augmentations which they contain. For geometric augmentations, we study and later use the MMPose implementation [51], and for the rest of the augmentations, we focus on the popular Python library Albumentations [5]. Hence we introduce the augmentations as they are implemented in these two toolboxes including their input parameters.

We group the augmentations into five groups: geometric, blur, color, noise, and weather augmentations. We notice that this grouping is more subtle than the usual division into brightness and geometric transformations. The augmentations grouping tries to group the augmentations according to their effect on the image and is done from our subjective perception of the resulting image. a good reason to create smaller groups from the transformations is an opportunity to further explore the effect of individual augmentations on training. One could argue that we could study the effect of each augmentation individually, but as we show in the following Section a lot of the augmentations work in a very similar way. Another reason to group the augmentations is to save computational time.

6.1 The augmentation groups

In this Section, we describe the augmentation groups and the augmentations they contain in more detail. We go over each augmentation and describe its nature and the parameters that need to be set. Most of the parameters for the Albumentations augmentations are given in a form of an interval from which the final parameter value is chosen randomly. Unless stated otherwise, the final parameter is chosen uniformly from the interval.

Geometric augmentations

There are four truly geometric transformations in the group: the horizontal flip, random scale, random rotation, and random shift. The last augmentation, half-body transformation, is not geometric by nature and augments the ground-truth target. We later use it along with the geometric transformations, so include it in this group. All of the geometric augmentations also transform the ground truth keypoints to correspond to the augmented image.

The horizontal flip flips the image horizontally.

The random scale changes the image size by a factor s passed as the input parameter.

The random rotation rotates the image by an angle θ passed as the input parameter.

The random shift implementation we choose shifts the bounding box center. The center is shifted both horizontally and vertically and the new position of the center $(x'_c, y'_c)^T$ is given as

$$\begin{bmatrix} x'_c \\ y'_c \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} + \begin{bmatrix} s_h \cdot w \\ s_v \cdot h \end{bmatrix}, \tag{19}$$

where $(x_c, y_c)^T$ is the original bounding box center, w and h are the bounding box width and height and s_h and s_v are the shift factors passed as input parameters. The shift of the bounding box center is not a geometric transformation per se, but in the top-down pose estimation task the model input is only the content inside the bounding box, hence the shift combined with this crop becomes a standard shift transformation.

The last augmentation is the half-body transformation, which keeps only all the upper or all the lower body keypoints. The differentiation to lower and upper body keypoints for our Lynx-Pose dataset is adopted from the Animal-Pose dataset. The lower body keypoints are the tail-base and all of the back limb keypoints, the upper body keypoints are withers, the throat, the front limb keypoints, and the head keypoints. This augmentation is performed only when the number of visible keypoints is above a threshold t passed as the input parameter. In Section 5.1, we have shown that most of the images in our dataset have all of the keypoints. This augmentation tries to "teach" the model, that it does not always need to produce high confidence for all of the keypoints. Since the keypoints can often be occluded, this is a really important ability to have and the network needs to know how to represent this possibility.

Blur augmentations

The blur augmentations group consists of four transformations that blur the image: averaging blur, gaussian blur, median blur, and motion blur. All of these blurs have the input parameter k, which is the blur mask size. The Albumentation implementation of these augmentations only accepts an odd mask size. An example of blur augmentations of an image from our Lynx-Pose dataset is in Figure 21.

The averaging blur convolves the image with a mask of a given size. The mask is a matrix of ones divided by the total number of its elements.

The Gaussian blur takes a Gaussian kernel of a size k and convolves it with the image. The standard deviation σ of the Gaussian kernel is calculated as $\sigma = 0.3(0.5(k-1)-1)$.

The median blur replaces each image pixel with the median of neighboring pixel values. The neighboring pixels are taken from a square neighborhood centered at the currently replaced pixel and the size of the square side is k.

The motion blur tries to simulate the motion of the objects in the image. It convolves the image with a mask with values only in a line and the mask elements sum up to one. The direction of this line in the mask matrix is the direction of the movement. For arbitrary angles, the best "approximation" of a line in the given direction is in the mask. An example of such masks for vertical, respectively horizontal, respectively 45° movement direction is as follows:

$$\begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}, \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}^T, \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix}.$$

In Albumentations, only a horizontal direction of the movement is used.

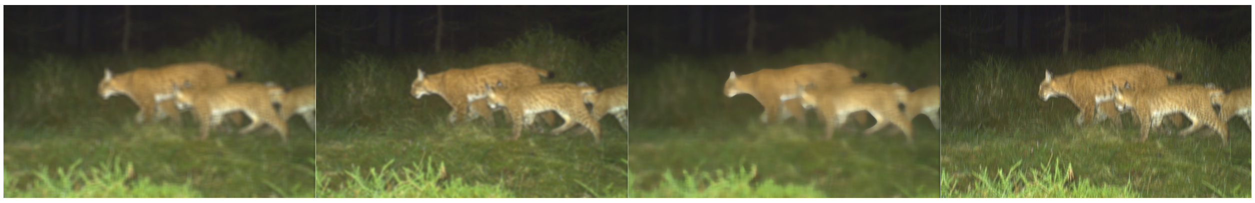


Figure 21: Example of blur augmentations applied on the image (Left to right: averaging blur, Gaussian blur, median blur, motion blur).

Color augmentations

The color augmentations group is a group of brightness transformations that globally change the brightness values of the color channels. This group contains 7 augmentations: channel dropout, channel shuffle, color jitter, hue-saturation-value transformation, image inversion, RGB shift, and random change of brightness and contrast. Examples of color augmentations of an image from our Lynx-Pose dataset are shown in Figures 22 and 23.

The channel dropout augmentation replaces the brightness value in a random channel with a value v selected as the input parameter.

The channel shuffle shuffles the channels randomly and takes no parameters.

The color jitter augmentation changes the brightness, contrast, saturation, and hue values randomly. The new image I' is calculated from the old image I as follows:

$$I_b = k_b \cdot I, \tag{20}$$
$$I_c = k_c \cdot I_b + (1 - k_c)\,\bar{G}_b, \tag{21}$$
$$I_s = k_s \cdot I_c + (1 - k_s)\,G_c, \tag{22}$$
$$H' = (H_s + 360 k_h) \mod 360, \tag{23}$$

where I_b, I_c, and I_s are the images after changing brightness, contrast, and saturation. The grayscale image of the image I_c is denoted as G_c and $\bar{G}_b$ is the mean of the grayscale image for the image I_b. H' and H_s denote the hue channel in the HSV image representation of the respective images I' and I_s. The constants k_b, k_c, k_s, and k_h are the input parameters.

Hue, saturation, and value (HSV) transformation is very similar to color jitter. It works fully in the HSV image representation. Unlike in color jitter, the changes k_h, k_s, and k_v in hue, saturation, and value are additive. These changes are the transformation's input parameters.

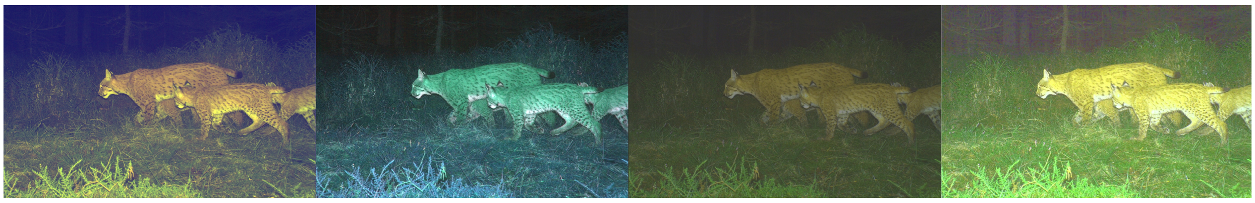


Figure 22: Example of color augmentations applied on the image (Left to right: channel dropout, channel shuffle, color jitter, HSV transformation).

Image inversion inverts the input image by subtracting its brightness values from the maximum value of the image type. This augmentation takes no input parameter.

The RGB shift augmentation adds a value to or substracts a value from all pixels within a given color channel. This is done for each channel and the values k_r, k_g, and k_b added to the color channel pixels are the input parameters.

The last transformation in this group is the random change of brightness and contrast. The new image I' is generated from the old image I as

$$I' = k_b \cdot I + k_c \cdot m, \tag{24}$$

where m is the maximum value of the image type and constants k_b and k_c are the input parameters.

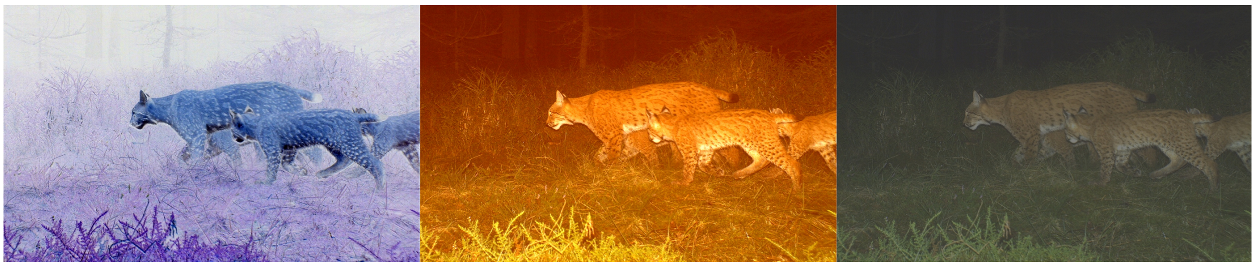


Figure 23: Example of color augmentations applied on the image (Left to right: image inversion, RGB shift, random brightness and contrast).

Noise augmentations

The noise augmentations add noise to the image or reduce the image quality by some other means. We have chosen the following 6 augmentations for this group: the downscale transformation, additive Gaussian noise, ISO noise, multiplicative noise, posterize transformation, and sharpening. Examples of noise augmentations of an image from our Lynx-Pose dataset are shown in Figures 24 and 25.

The downscale transformation is considered a geometric transformation, but in the Albumentations implementation, the augmentation first downscales the image to a lower scale and then upscales it back to the original resolution. This means that there is no change in the image geometry and the noise brought to the image is introduced by the interpolation algorithms. The augmentation uses the nearest neighbor interpolation as the default interpolation, other methods like bilinear or bicubic interpolation can be chosen as the input parameter. The scale value s to which the image is downscaled is the input parameter.

The additive Gaussian noise adds a noise with the Gaussian probability density function to the image. The noise can be the same for all the channels or generated for each channel separately, this setting is passed at the input of the method. The Gaussian distribution's variance σ^2 and the mean μ are also the input parameters.

The ISO noise adds a simulated grain noise which appears naturally in the images when the camera ISO value is set too high. In photography, the ISO value determines the camera's sensitivity to the light. However, this step is done in the post-processing and hence amplifies both the brightness values and the random noise present in the incoming signal. The augmentation is applied to the hue, luminance, and saturation (HLS) image representation. The noisy luminance L' is calculated from the old luminance L as

$$L' = L + (1 - L)\frac{P}{255}, \tag{25}$$

where P is Poisson noise with mean $\lambda = \sigma \cdot i \cdot 255$. The parameter σ is the standard deviation of the HLS values and i is the intensity given as the input parameter. The noisy hue H' is calculated from the old hue H as

$$H' = H + G, \tag{26}$$

where G is Gaussian noise with zero mean and variance $\sigma_G^2 = c \cdot 360 \cdot i$, parameter c is also the input parameter.

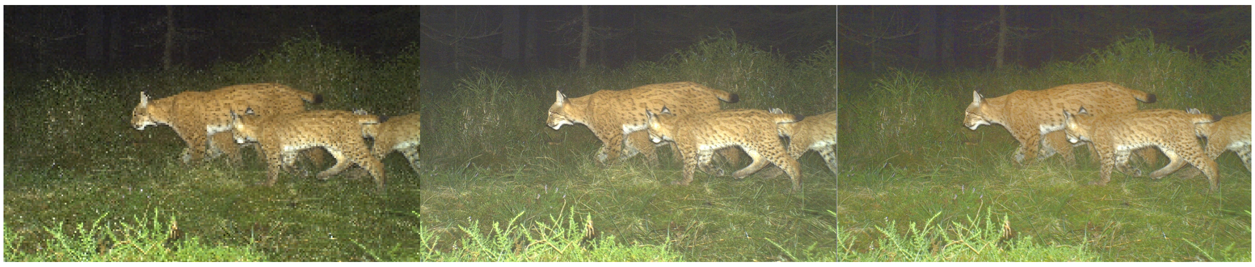


Figure 24: Example of noise augmentations applied on the image (Left to right: downscale, additive Gaussian noise, ISO noise).

The multiplicative noise multiplies the image by a given scalar value or does an element-wise multiplication by a matrix. This can be chosen as the input parameter of the augmentation. Another choice is to either have the same value(s) for all channels or a different one(s). The interval k_m from which the multiplier value(s) is (are) generated is given as the input parameter.

The posterize transformation reduces the number of bits used to represent the values of each color channel. This augmentation takes the image with the 8-bit unsigned integer number representation format and randomly chooses a new maximal number of bits to represent each channel. These maximal numbers k_r, k_g, and k_b are given as the input parameter for each channel.

The last augmentation in the noise group is the sharpen augmentation. This augmentation convolves the image with the 3×3 sharpening matrix $\boldsymbol{S}$. The base sharpening matrix is given as

$$\boldsymbol{S} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}. \tag{27}$$

The sharpen augmentation in Albumentations takes parameter α and a parameter l for lightness as its input. These parameters are then used to generate the final sharpening matrix ***S'*** as follows:

$$\boldsymbol{S'} = (1-\alpha)\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \alpha \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8+l & -1 \\ -1 & -1 & -1 \end{bmatrix}. \tag{28}$$

The matrix ***S'*** is then convolved with the image.

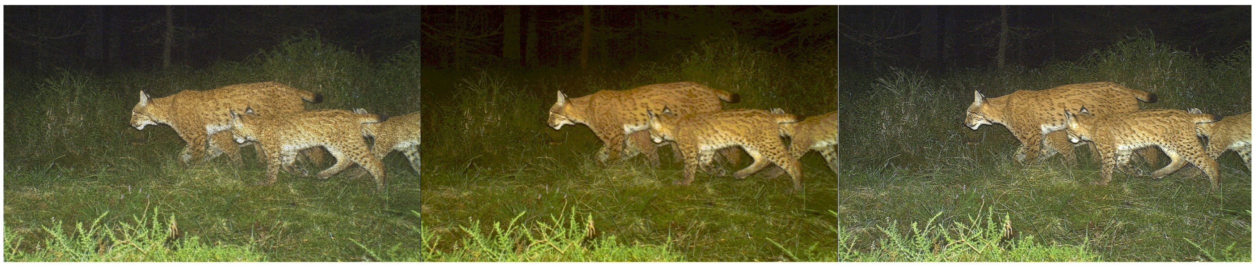


Figure 25: Example of noise augmentations applied on the image (Left to right: multiplicative noise, posterize, sharpen).

Weather augmentations

The last augmentation group is augmentations simulating various weather effects on the environment in the image. There are four augmentations in this group: random snow, random sun flare, random rain, and random fog. An example of weather augmentations of an image from our Lynx-Pose dataset is in Figure 26.

The random snow bleaches out the values of some pixels to add a snow effect to the image. This augmentation uses a given brightness coefficient k_b which is used to multiply the values below a given threshold t. The condition set by the threshold is checked against the values of luminance in the HLS image representation. Both the coefficient and the threshold are the input parameters.

The random sun flare imitates an effect of contre-jour images where the camera is pointing toward the source of the light. This generates a number of circles with a set color and adds to the image with increasing transparency. The color c, number of circles n, and radius of the first generated circle r is the input parameter of the augmentation.

The random rain adds lines to the image simulating the water drops during rain. The lines are generated with a given slant (rain angle) same for all raindrops. After the drops are generated and added to the image, the image brightness is reduced by a brightness coefficient $k_b < 1$ and then it is also blurred by an averaging blur. The rain slant θ, the size of the blur mask k, the brightness coefficient k_b, and the color of the drops c are the input parameters. The rain type is also chosen from “drizzle”, “heavy”, and “torrential”. The type influences the generated lines’ length and the number of drops.

The last weather augmentation is random fog which simulates foggy weather conditions. The fog is simulated by creating circles and overlaying them with the image at the given points. The number of circles is calculated using the fog coefficient k_f, which is passed as the input parameter. The circles are then overlayed with a weight given by a coefficient α also passed as the input parameter. The image is then blurred using averaging blurring with kernel size calculated from the fog coefficient.

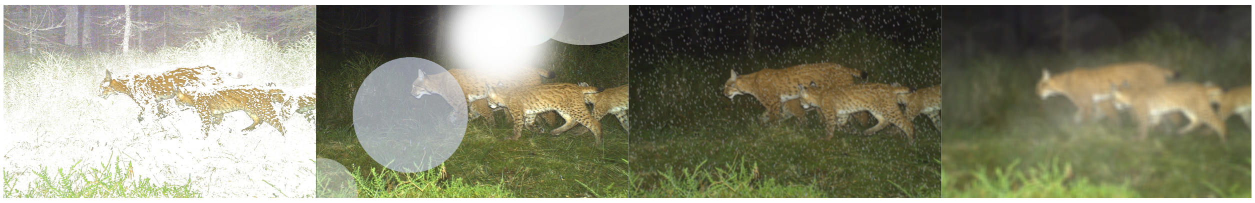


Figure 26: Example of weather augmentations applied on the image (Left to right: random snow, random sun flare, random rain, random fog).

7 Experiments

In this Section, we present the models chosen for the training and the results of various experiments with the training. First, we present the experimental settings, then we evaluate the performance of the models on our data without any training. Afterward, we train the models with our basic training settings and do a study on the effect of image augmentations on the training result. At last, we try to train a model with a randomly initialized keypoint head and compare its performance to the pre-trained heatmap-based models.

7.1 Experimental settings

We use the MMPose [51] framework for our experiments which features a model zoo with pre-trained models. From the available models, we choose HRNet-W32, ResNet-50, and ResNet-151 backbones with a pre-trained heatmap head and the input size 256×256. For our experiments, we choose models pre-trained on the Animal-Pose dataset, which was introduced in Section 5.2. The main reason for this choice is that the keypoints used to represent pose in the Animal-Pose dataset are the same 20 keypoints our task uses, hence we can use the pre-trained weights for the heatmap head. We show the importance of pre-training later in Section 7.4. The chosen Animal-Pose dataset also focuses on quadruped mammals that have similar body composition to that of lynx. One of the species in the dataset is cats, which also makes a good basis for lynx pose estimation. These models have achieved mAP of 0.736, 0.688, and 0.709 on the validation set of the Animal-Pose dataset.

As mentioned in Section 5.1, our dataset contains 1685 images of lynxes in various environments. For our experiments, we use a 75-25 train-validation split of the data meaning we have 1262 images in our train set and 423 images in our validation set. We convert the annotations to the Animal-Pose annotation format and use the data loader created for the Animal-Pose dataset. After the image is loaded by the data loader, it is cropped using the bounding box and rescaled to the network input size of (256×256 pixels).

We use the MMPose implementation of TopdownHeatmapSimpleHead as our heatmap prediction head and as mentioned above, this prediction head is also pre-trained on the Animal-Pose dataset. We presented this head in Section 3.1, and in Section 3.3. It is the head composed of 20 convolutional layer channels with 1×1 kernel sizes for 20 heatmaps. The heatmap count corresponds to the number of predicted keypoints. The final keypoint position $(\hat{x}', \hat{y}')$ is calculated as described in Equation 3 in Section 3.3. This heatmap head from MMPose also produces a confidence score for keypoints. This confidence score for a given keypoint is the maximal value in a heatmap corresponding to the keypoint.

The ground truth heatmaps are generated using the Gaussian approach also described in Section 3.3 with the standard deviation parameter set as $\sigma = 2$ and the mean equal to the ground truth keypoint coordinates.

The loss function for the training is the MSE loss presented in Section 3.3 as presented in Equation 6. The target weight w_k for our experiments is set as 1 if the ground truth keypoint is annotated and 0 if not.

As the main metric for the trained models evaluation and to choose the best model weights, we use average precision on the validation dataset. We defined this metric in Section 4.4 and we use the mAP calculated at OKS=0.50:0.05:0.95. The mAP calculation on the validation dataset is done after each training epoch.

In the experiments, we use the same settings of the following hyperparameters for all of the runs: the batch size is always set to 32 images per batch, the initial learning rate is always set to $5e^{-4}$, the warmup is linear and is performed over 5 epochs. The learning rate value at the start of the warmup is $5e^{-8}$, i.e. $1e^{-4}$ times the initial learning rate. The linear change during warmup happens in each training step. The step scheduler is used and has a learning rate decay factor of 0.3. The momentum for the optimizer is set to 0.9.

We use the Adam optimizer for all experiments. We tried using the SGD optimizer with various hyperparameter settings. However, setting it correctly is difficult and in our experiments, the model did not achieve comparable results to using Adam and converged much slower. One such run is depicted in Figure 27. The training run with the Adam optimizer runs 110 epochs, and SGD was set to 280 epochs. We see that the run with Adam converges in the 100th epoch and reaches a maximum mAP value of 0.9113. For SGD, on the other hand, we notice the trend of the mAP value increasing slightly even after this 100th epoch, reaching a maximum of 0.8838 in epoch 210.

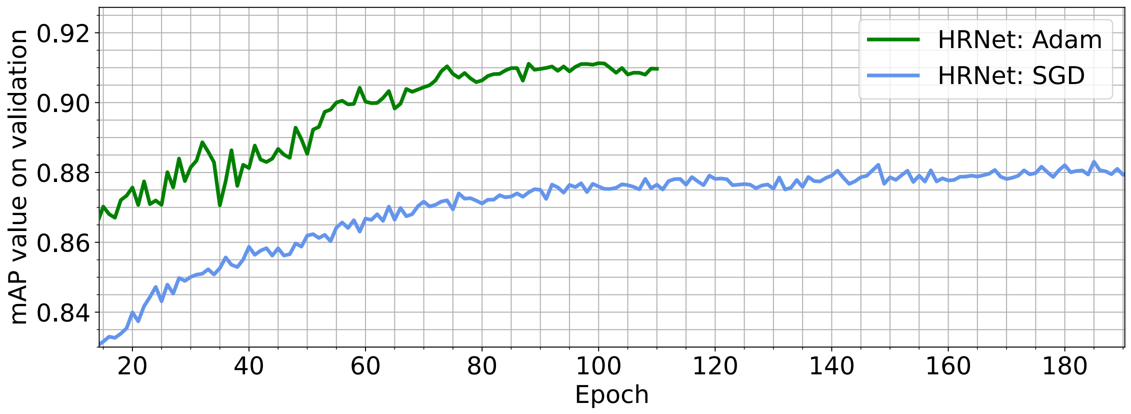


Figure 27: Comparison of mAP value on validation in a fine-tuning experiment run with SGD and Adam optimizers.

7.2 Baseline results

First, we calculate the mAP value for the selected models without any finetuning. The value is calculated on the Lynx-Pose validation dataset. The results can be seen in Table 1, comparing these values to the results the models have on the Animal-Pose dataset, we see that HRNet outperforms the two other models in both cases. The difference in performance between Animal-Pose HRNet and Resnet-50 is 0.048, the difference in performance is more than twice as big for the baseline Lynx-Pose models - 0.1112. The difference between the baseline Lynx-Pose HRNet and Resnet-152 is 0.0588.

	Mean Average Precision
HRNet	**0.5918**
ResNet-50	0.4807
ResNet-152	0.5330

Table 1: Mean Average Precision of the selected models without finetuning on the validation dataset.

After testing the baseline models on the validation set, the models are trained on the training dataset. For the first experiment, we do not augment the training images in any way during the run. We also run a second training experiment set where the images in the training dataset are augmented using a horizontal flip with a 50% probability. Apart from this augmentation, no other augmentation is applied to the training dataset images.

The hyperparameters for setting the scheduler steps and the number of epochs were set experimentally based on previous experiments. The other hyperparameters are set according to Section 7.1. For HRNet and ResNet-50, the epoch count and the scheduler steps settings are identical. The models are trained for 160 epochs and scheduler steps occur at epochs 50, 90, and 130. At each step, the learning rate is reduced by a factor of 0.3, the final learning rate is $1.35e^{-5}$. For ResNet-152, due to the larger model size, we choose a larger epoch count. For this experiment, we set the count at 180 epochs and the scheduler steps happen at epochs 60, 110, and 150. The learning rate reduction factor is the same for ResNet-152.

First, we compare the effect of the flip transform on the training results. This comparison of the best mAP values achieved by the models is shown in Table 2. We notice that the flip augmentation helped to increase the best mAP value for all of the models. ResNet-152 has the least increase in performance and both ResNet-50 and HRNet have a similar increase in performance. We also see that training the models increased their performances by approximately 35% compared to the baseline results presented in Table 1. Again, HRNet achieves the best mAP value results in this experiment. ResNet-50 achieves a result that is a tenth of a percent worse in both experiments.

	mAP without augmentations	mAP with flip
HRNet	**0.8893**	**0.8999**
ResNet-50	0.8802	0.8884
ResNet-152	0.8906	0.8946

Table 2: The best mAP values achieved by models on validation dataset with no augmentations and with flip augmentation being used during training.

The mAP value progression on the validation dataset during training is shown in Figure 28. We can now compare the epoch of convergence to the best mAP values between the two training experiments. HRNet with flip augmentation achieves its best mAP value in the 58th epoch. Without the flip augmentation, the mAP of 0.8893 is achieved in the 104th epoch which is almost twice the time. With the mAP value of 0.8946, ResNet-152 with flip augmentations achieves comparable results even faster than HRNet in epoch 40. Without the flip, the maximal value of 0.8906 is achieved in epoch 158, which is almost four times longer. ResNet-50 reaches its best mAP value of 0.8884 in the 95th epoch with flip transform, it is the exception in the experiment reaching the best mAP value without the flip in the 71st epoch.

This faster convergence of the models with a flip augmentation is could be caused by the models not overfitting to the training data as soon as the training runs without the augmentations. This gave the models a chance at learning to estimate a general lynx pose at the early stages of the runs. We see the mAP value actually decreasing for the flip experiments as the epochs progressed and the model started overfitting itself.

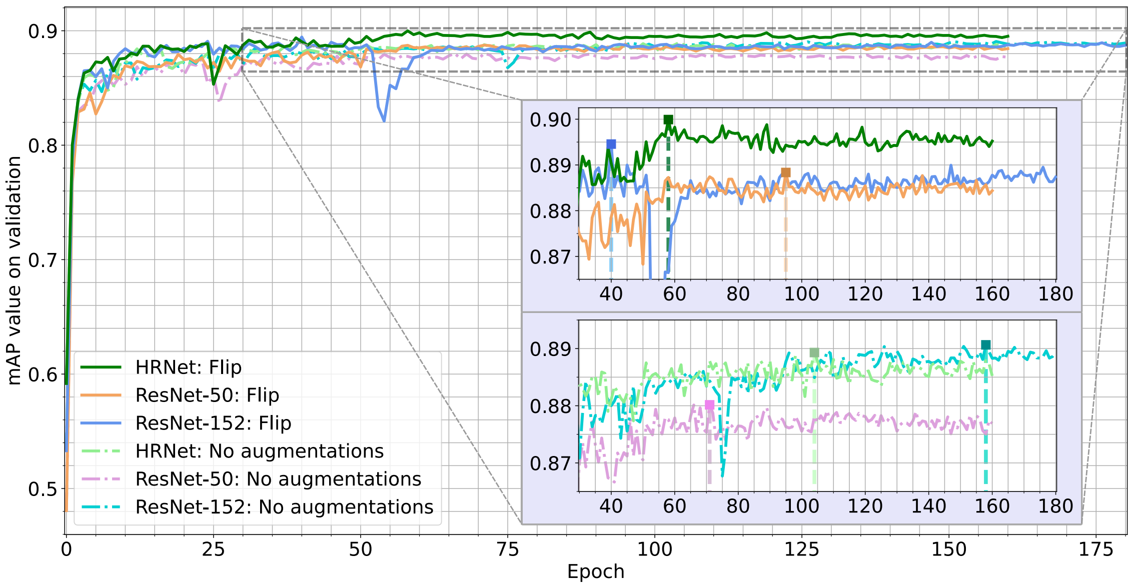


Figure 28: The comparison of mAP progression on the validation dataset with runs without any training data augmentations and runs with flip augmentation.

7.3 Augmentation experiments

In this Section, we experiment with the effect of the augmentation groups presented in Section 6 on the trained model performance. We run each experiment only once for each model as our goal here is to find the best model and the study of the augmentation effect on training is the side product. Hence, we run each experiment only once and the statistical significance of the conclusions we make is not computed.

We use the augmentations online, meaning we generate new augmented images for each training batch. For each augmentation, we set its application probability, we present this setting later in Section 7.3.1. In these experiments, we always use geometric augmentations and one additional group of presented augmentations. The geometric augmentations usage is to prevent the model from overfitting to the known joint locations. As we saw in the previous Section, adding the flip transformation already helped with achieving a better performance.

As we presented in Section 6, the augmentations take a lot of input parameters. Most of these parameters are given as an interval from which the value is chosen randomly. Since the parameter count is this high, we decide to choose these parameters empirically based on the perceived effect of augmentations on the images. Even this way, we have to manually set 72 parameters and check the resulting images. As we saw in Section 6, some of the augmentations have up to four interval parameters. This means we have to manually check 16 interval boundary combinations to verify the images are still usable for the training. We observe that an experimental search for the optimal parameter setting would be extremely computationally intensive. The whole models would need to be trained with fixed augmentation settings for tens, hundreds, or even thousands of runs while trying to optimize hyperparameters in a 72-dimensional space. Another question to consider is the difference in the performance between the optimal and the empirical parameter settings. Hence we settle for an empirical setting of the input parameters.

7.3.1 Training settings

Let us now discuss the parameters used for the experiments. First, we discuss the general training settings and afterward, we focus on the settings of the augmentations parameters.

For the experiments presented in this Section, we use the same dataset, models, losses, evaluation metric, and hyperparameters settings as in Section 7.1. The only change in the experiments is the scheduler policy. We use the same scheduler settings for HRNet, ResNet-50, and ResNet-152 across the experiments presented in this Section. The HRNet is trained for 180 epochs with a starting learning rate of $5e^{-4}$. The step scheduler is used with a learning rate decay rate of 0.3 and the steps are set to happen at the 45th, 100th, and 140th epoch, the learning rate first drops to $1.5e^{-4}$, then to $4.5e^{-5}$ and finally to $1.35e^{-5}$. Exactly the same setting is applied to the scheduler for ResNet-50. ResNet-152 is trained for 190 epochs with the same initial learning rate and the same decay. The scheduler steps for ResNet-152 happen at the 50th, 110th, and 160th epoch and the learning rates throughout the training have the same values as discussed before.

Let us now look at the settings of the data augmentation probabilities and the training data preparation pipeline. First, we use the bounding box shift with the probability of 30% and then flip the image horizontally with the probability of 50%. Then we use the augmentation group for which the experiment is being done - blur, color, noise, or weather augmentation. Only one augmentation from the group is selected and then it is applied with a probability of 80%. After, we apply the half-body transformation with the probability of 30% if the image has more than 8 keypoints. After that, the image is always transformed by random scale and rotation. The bounding box region of the image is then cropped and rescaled to the network input size, normalized to the model's expected mean and standard deviation, and sent to the network input. This training data generation process is the same for all of the presented experiments.

The parameter settings are listed in Appendix A. As stated earlier, a lot of these settings are given as intervals. The parameter choice from this interval is always done uniformly in the MMPose and Albumentations implementations.

7.3.2 Results

In this Section, we present the training results along with the mean average precision progression visualizations on the validation dataset. First, we focus on the experiments with the different augmentation groups separately, then we compare the final results together.

Let us repeat that the only thing changing throughout the experiments is the augmentation group from which the augmentation is chosen after the horizontal flip is done. Everything else is set up as stated in Section 7.3.1.

The first experiment is done for the blur augmentation group. We see the mAP measured on the validation dataset after each training epoch in Figure 29. ResNet-152 achieves its peak mAP of 0.9020 first in 117 epochs. After that, ResNet-50 achieves mAP 0.8992 in 131 epochs. HRNet achieves the best mAP value of 0.9042 in this experiment in the 167th epoch. We can notice the increase in performance of ResNet-50 on the 100th epoch, where the learning rate changes to the value of $4.5e^{-5}$. Both ResNet models reached the best mAP value before the last scheduler step. Only HRNet reached its best value after the last scheduler step. We also see the progression being very intertwined before the last scheduler step. After this step, HRNet starts overperforming the two other models.

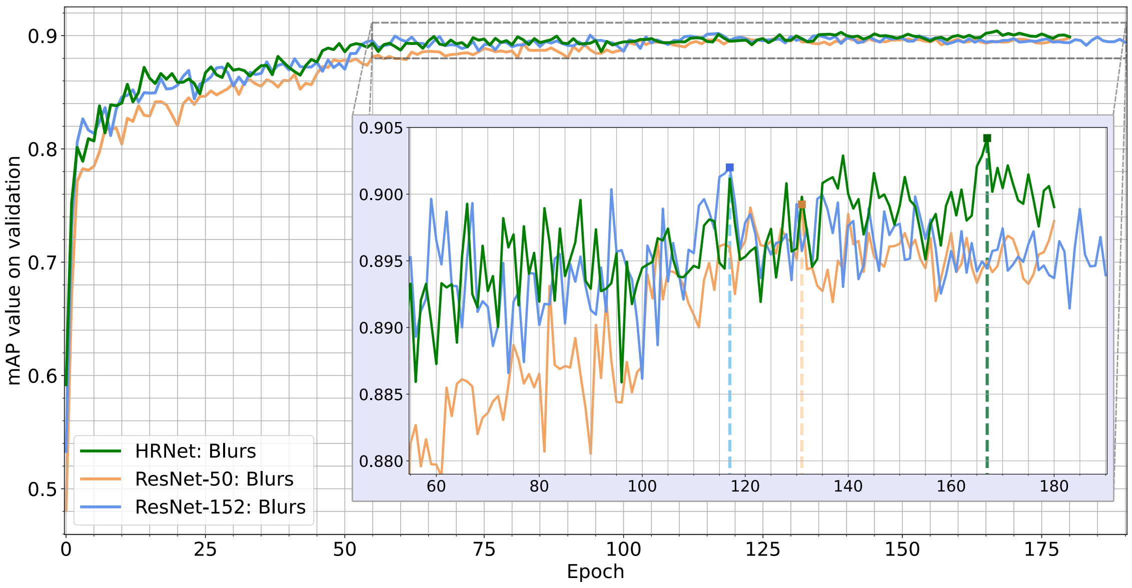


Figure 29: The comparison of validation mAP progression of the runs trained with the blur augmentations in the training image preparation pipeline.

The second experiment is conducted with the color augmentation group. The mAP value progression on the validation dataset is in Figure 30. ResNet-152 achieves its best mAP of 0.9134 in epoch 120. ResNet-50 is the slowest network to reach its best mAP value achieving 0.9091 in epoch 165. HRNet is in turn the fastest here achieving mAP value of 0.9171 in epoch 104. Comparing the progression to the one in Figure 29, we see that the results for individual networks are less intertwined in the later training stages. We also notice a similar increase in the performance of the trained models after the penultimate scheduler step as we did in the last experiment. In this experiment, the HRNet and ResNet-152 seem to be affected more by this scheduler step.

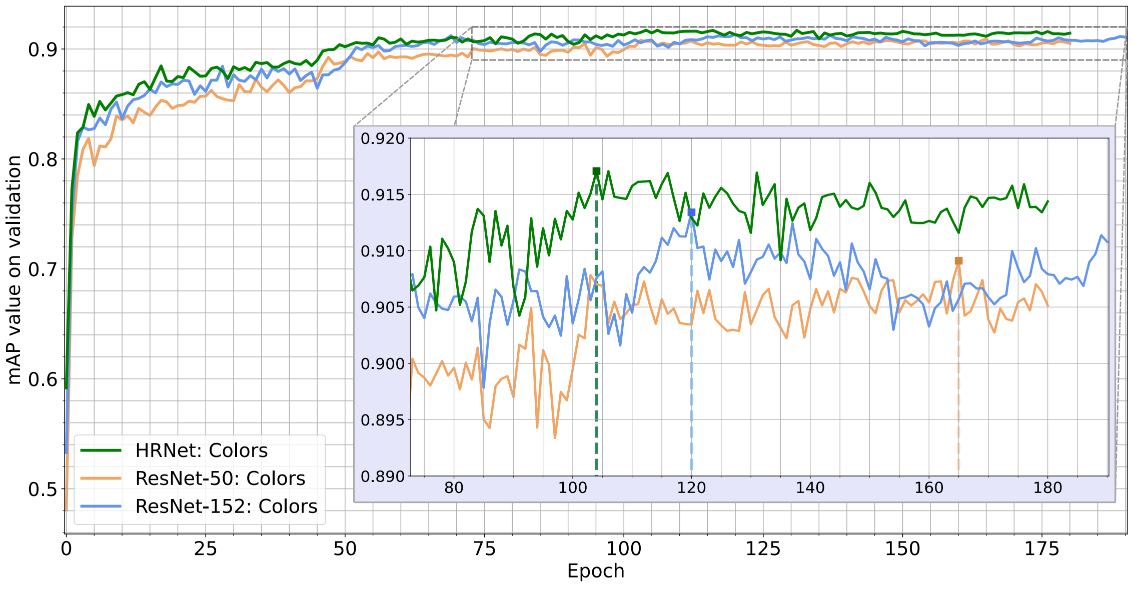


Figure 30: The comparison of validation mAP progression of the runs trained with the color augmentations in the training image preparation pipeline.

The third experiment is done with the noise augmentation group. The mAP measured on the validation dataset for this experiment is depicted in Figure 31. ResNet-152 achieves its best mAP value of 0.9115 in epoch 136 and is the fastest network in this experiment. ResNet-50 has its best mAP value of 0.9079 in epoch 144. HRNet is again the “slowest” architecture achieving the best mAP value of 0.9157 in epoch 175. All of the networks achieved their respective best mAP values later than in the previous experiments. The maximal values for HRNet and ResNet-50 were achieved after the last scheduler step. ResNet-152 reached its best performance before the last step. We also see both ResNet-152 and HRNet overperforming ResNet-50 throughout the experiment. This could mean that ResNet-50 does not have the capacity to respond to the noises as well as ResNet-152, because we saw the two models having a similar mAP progression in the previous experiments.

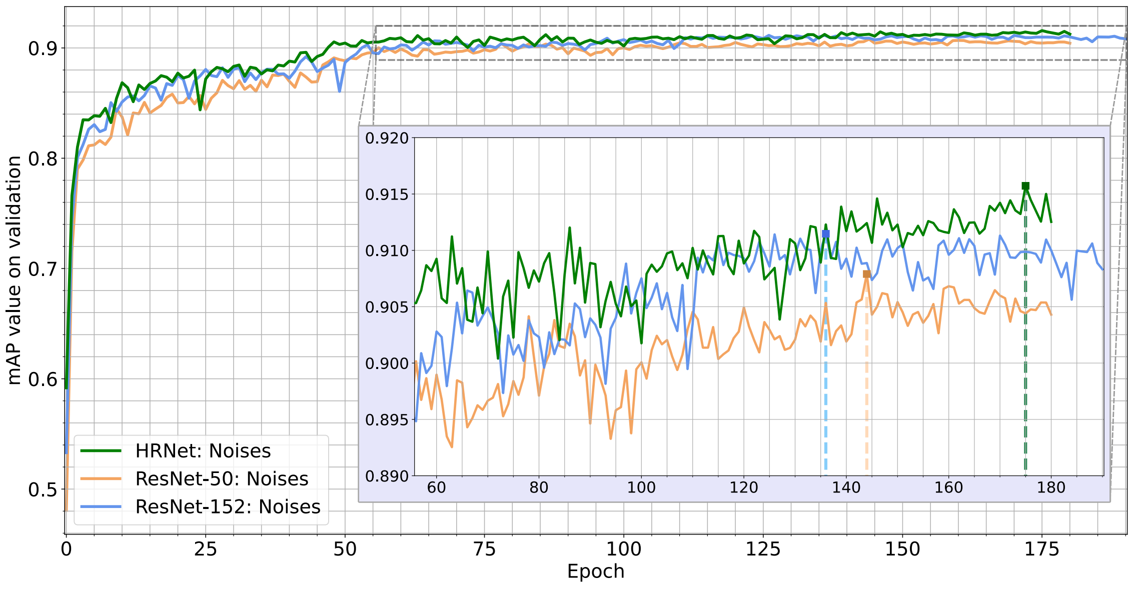


Figure 31: The comparison of validation mAP progression of the runs trained with the noise augmentations in the training image preparation pipeline.

The fourth experiment is done with the weather augmentation group. This experiment is depicted in Figure 32. Again, the networks achieve their best mAP values later than in the first two experiments. ResNet-152 achieves its best mAP of 0.9089 in epoch 150. ResNet-50 achieves its best mAP of 0.9019 in epoch 178, being the “slowest” network in this experiment. HRNet achieves the best mAP value of 0.9140 in epoch 149, beating ResNet-152 by one epoch. This experiment is interesting with regard to the relations of the individual mAP progressions. In the previous experiments, the models’ mAP values were somewhat intertwined in the plots, especially for the blur and noise augmentation groups. Here we see, that the HRNet’s mAP value is better than the one of the ResNet-152 in the majority of the epochs. The same can be said for ResNet-152 and ResNet-50. We assume HRNet has a better response to weather augmentations, but more runs would be required to validate the assumption. This better response could be caused by the ability of the model to keep a high-resolution representation of the image. As the augmentations bring a lot of noise that is not filtered easily and introduce new edges in the images (e.g. the raindrops from random rain augmentation), the low-resolution representation of the ResNet architectures could be too coarse.

Another thing to consider with weather augmentations is the computational demand for fog augmentation. During the experiments, we notice that the weather augmentations experiment runs considerably longer than the other experiments. Hence we take the augmentations in the group and compare their computation time to selected augmentations from the other groups, namely color jitter for the color group, ISO noise for the noise group, and median blur for the blur group. We see the results in Table 3, where we notice the fog augmentation being 44 times slower than the fastest augmentation - color jitter. We also notice the fog augmentation is 9 times slower than the second slowest weather augmentation - the random sun flare denoted as "Sun".

This is given by the big input image size (2048 × 1536px). We also include the relative computation time of the augmentations on a small image (373 × 205px) in the same Table. All of the augmentations run at comparable speeds after cropping the images to a smaller size. Hence the fog augmentation speed is no longer an issue and if we wanted to experiment with the weather augmentations more, we would consider applying them after the input is already cropped by its bounding box.

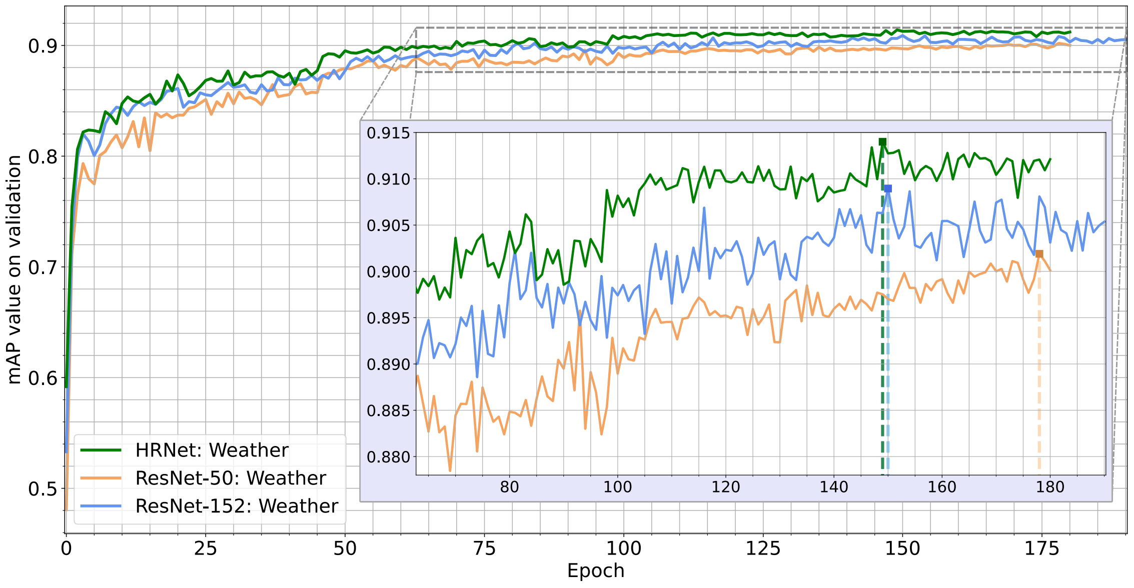


Figure 32: The comparison of validation mAP progression of the runs trained with the weather augmentations in the training image preparation pipeline.

Augmentation (Big image)	Color	Noise	Blur	Fog	Rain	Snow	Sun
Relative time (to Color time)	1×	13×	8×	45×	3×	2×	5×
Augmentation (Small image)	Color	Noise	Blur	Fog	Rain	Snow	Sun
Relative time (to Color time)	0.03×	0.27×	0.19×	0.06×	0.05×	0.04×	0.14×

Table 3: The comparison of the times taken to augment the image relative to the color jitter augmentation on the original image.

Having done the experiments with the basic augmentation groups, we found the color and the noise augmentations groups to have the best effect on the models' performance. We also notice that the blur group has very little effect on the performance and the same can be said for the weather group in regard to the two ResNet models. Hence we now train the models with color augmentations followed by noise augmentations.

The augmentation group used in the training data generation pipeline is now replaced by these two groups. In each image preparation step, one augmentation from each group is selected randomly. These two selected augmentations are then applied each with 80% probability (meaning that the probability of both being applied is 64%).

The first experiment combining the augmentation groups is applying color augmentation first followed by noise augmentation. The mAP value progression on the validation dataset in this experiment can be seen in Figure 33. ResNet-152 reaches its best mAP value of 0.9099 in epoch 130. ResNet-50 is the last model to reach its best mAP of 0.9080 in epoch 167. HRNet reaches the best mAP value of 0.9110 in this experiment in epoch 158. Once again, the networks reach their best performance in the later training stages and ResNet-152 is the only network reaching its best mAP value before the last scheduler step. Interestingly enough, ResNet-50 is the only network seeing an increase in its performance compared to the previous experiments. ResNet-152 and HRNet on the other hand drop in performance compared to the experiments where the color and the noise groups were applied separately.

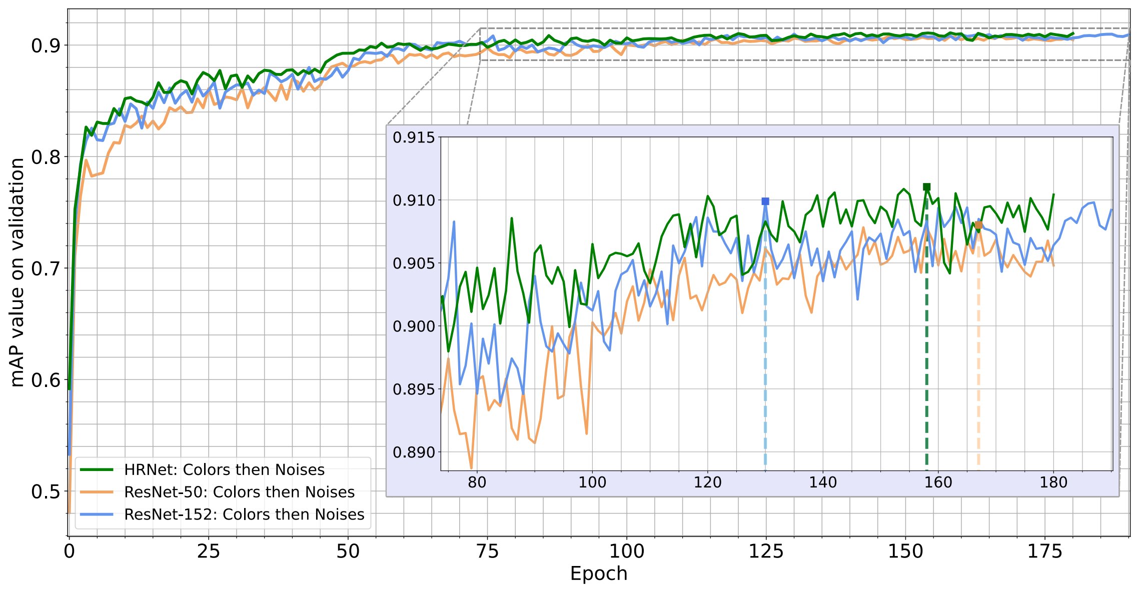


Figure 33: The comparison of validation mAP progression of the runs trained with the color and then noise augmentations in the training image preparation pipeline.

The second combination is done by applying noise augmentation first and following it with color augmentation. The mAP value progression can be seen in Figure 34. ResNet-152 reaches the mAP value of 0.9142 in epoch 140. ResNet-50 is slightly faster, reaching the mAP value of 0.9021 in epoch 137. HRNet is the last network to reach the maximum with its best mAP value of 0.9122 in epoch 176. Again we notice a big increase in the models' performance after the scheduler step in epoch 100 (110 for ResNet-152). The progression in this experiment is interesting because ResNet-152 achieves better mAP on the validation dataset than HRNet. Their mAP value progressions are also very similar. ResNet-152 seems to be responding to this augmentation combination the best as it is the only model which sees an increase in performance compared to all the previous results. If we look at the mAP value progression of ResNet-50, we notice it performs considerably worse than the other models throughout the experiment. This is a very similar phenomenon to the one observed in the Weather experiment.

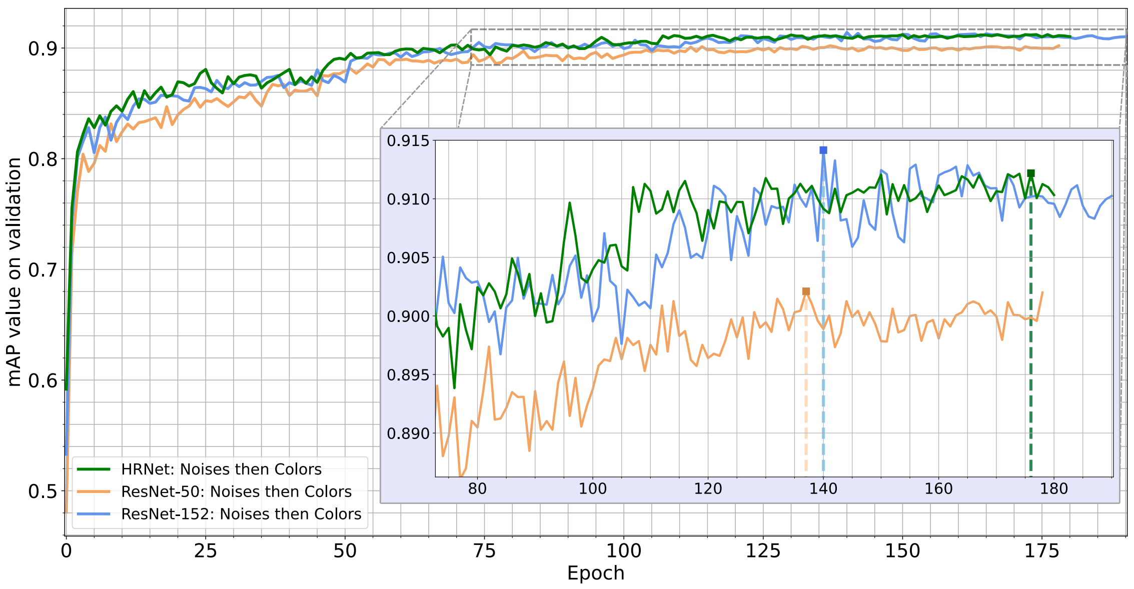


Figure 34: The comparison of validation mAP progression of the runs trained with the noise and then color augmentations in the training image preparation pipeline.

Since the augmentations combination brings only a slight increase in performance for ResNet-152 and not for the other two models, we look at the effect of the augmentations on the images. The hypothesis is that the combinations of the augmentations may decrease the image quality too much. We notice that some of the boundary parameter combinations produce images that are deformed beyond recognition. Hence we decide to tune the parameters for the groups, these parameters are also presented in Appendix A. We run a new training experiment for these tuned parameters.

We show the mAP value progression for this experiment run in Figure 35. We notice that all of the models reached their maximum mAP value late in the experiment. ResNet-152 reaches its best mAP value of 0.9119 in epoch 185. ResNet-50 reaches its best mAP value of 0.9069 in epoch 167. HRNet is the best-performing model in this experiment with mAP value of 0.9143 in epoch 175. The tuning did not really work for ResNet-152 and HRNet as the results are still comparable to the previous experiments. Both ResNet models achieved a better result in one or the other of the previous experiments. However, we notice ResNet-50 performance being much closer to the performance of the other models again. The HRNet's performance is better than in the previous experiments with augmentation combinations, but still worse than the one achieved in the color experiment where HRNet reached mAP value of 0.9171. Perhaps a longer training and another scheduler step would help with the performance since all of the networks achieved their best mAP values at the very end of the training.

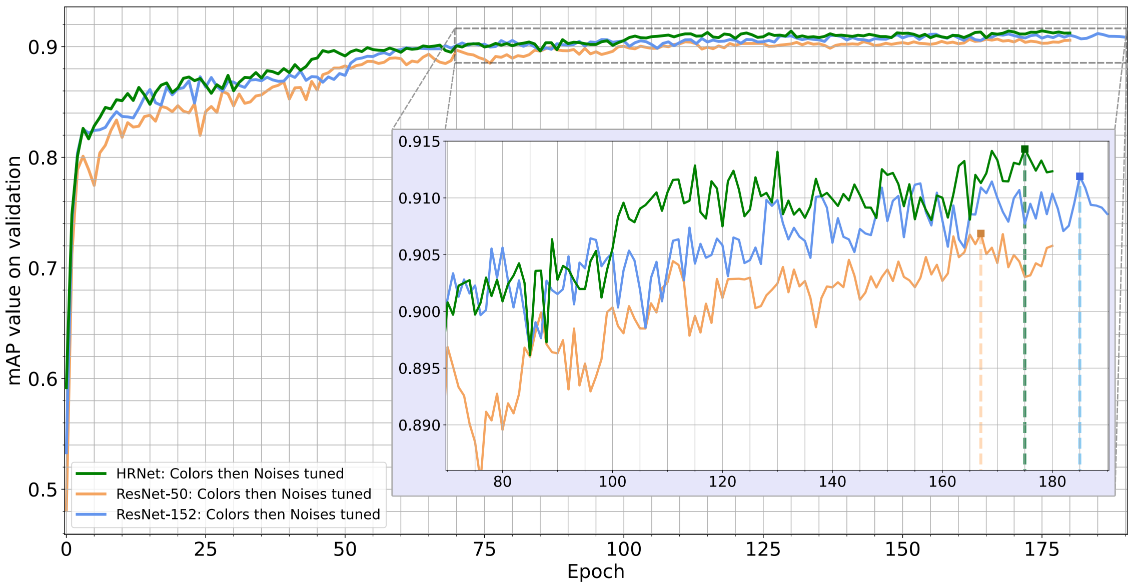


Figure 35: The comparison of validation mAP progression of the runs trained with the tuned color and then tuned noise augmentations in the training image preparation pipeline.

The last experiment we do with the models is combining the color and noise groups by applying either one or the other group in the pipeline. This is done by randomly selecting either the color or the noise group, then randomly picking one augmentation from the selected group, and applying it with the 80% probability. We also keep the initial settings for the groups and do not use the tuned ones from the last experiment.

We depict the mAP value progression for this experiment in Figure 36. ResNet-152 reaches its best mAP value of 0.9127 in epoch 115. ResNet-50 is the latest to reach its peak value reaching 0.9137 in epoch 177. HRNet also reaches its best mAP value of 0.9180 in the later stages in epoch 172. In this experiment, HRNet reaches the best mAP value of all the experiments, and looking at the Figure 36 we see it outperforming the two other models in the late training stages. Another quite interesting phenomenon is ResNet-50 getting better results than ResNet-152 and it also improved its previous best mAP value from the color experiments with an improvement of 0.0046. We see that this experimental setup probably helped to combine the best from the color and noise augmentation groups for ResNet-50 while alleviating the deformation of the images. The models reached their best performances at the very end of the training again. As stated in the previous experiment too, it would also be interesting to add additional epochs and scheduler step to try to increase their performance further.

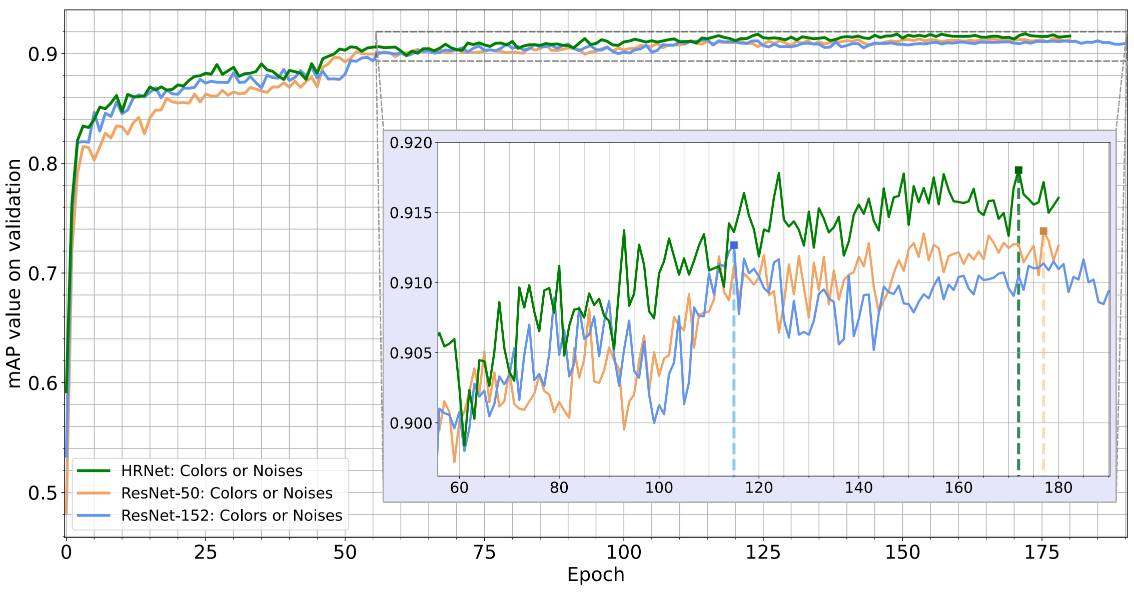


Figure 36: The comparison of validation mAP progression of the runs trained with the color or noise augmentations in the training image preparation pipeline.

We present all of the experiment results in Table 4. We highlight the best-performing models in each experiment with underlined text and the overall best result for a given model with bold text.

In the Table, we see that with the exception of two experiments, HRNet is the best-performing model. We also notice that ResNet-50 is the worst model, except in the last experiment where it outperforms ResNet-152. This best ResNet-50 achieved a better performance than five trained HRNet models and nine trained ResNet-152 models (excluding the baseline models). All of the best models are trained with the color and noise augmentations combination and the last experiment yielded the best results for both HRNet and ResNet-50.

	Baseline	No augmentations		Flip
HRNet	0.5918	0.8893		0.8999
ResNet-50	0.4807	0.8802		0.8884
ResNet-152	0.5330	0.8906		0.8946
	Blur	Color (C.)	Noise (N.)	Weather
HRNet	0.9042	0.9171	0.9157	0.9140
ResNet-50	0.8992	0.9091	0.9079	0.9019
ResNet-152	0.9020	0.9134	0.9114	0.9089
	C. then N.	N. then C.	C. then N. tuned	C. or N.
HRNet	0.9110	0.9122	0.9143	**0.9180**
ResNet-50	0.9080	0.9021	0.9069	**0.9137**
ResNet-152	0.9099	**0.9142**	0.9119	0.9127

Table 4: The best mAP values of the models achieved on validation dataset throughout the experiments.

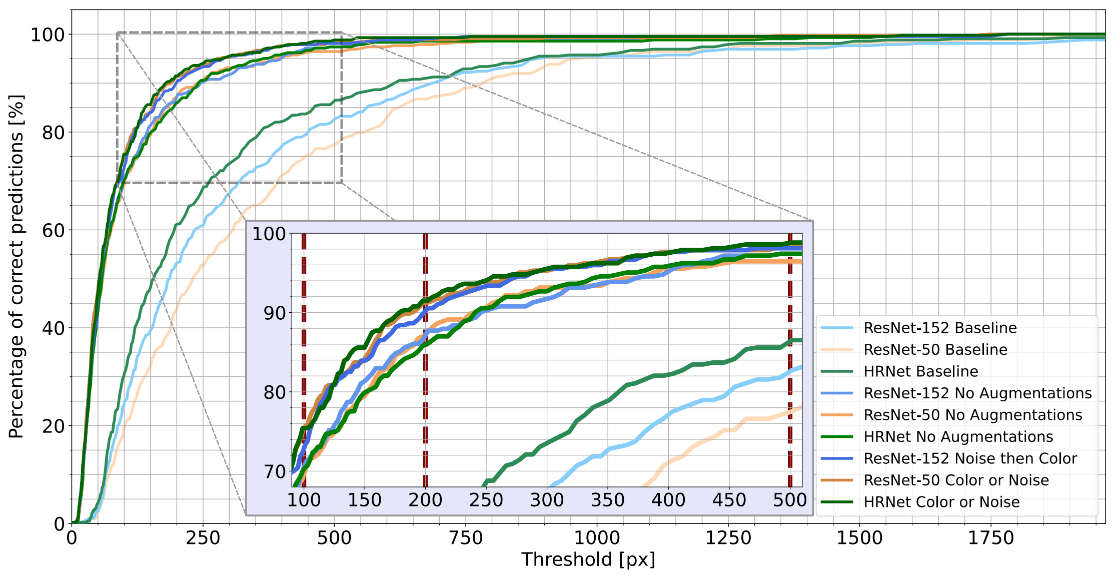


Figure 37: The percentages of correct predictions at given thresholds for the selected models.

To validate the obtained results, we compute the percentage of correct predictions metric which we introduced in Section 4.2. We compare the three baseline models with the models trained without the use of augmentations and with the best models from Table 4. The calculation is done on thresholds from 0.9 to 1960 pixels. Euclidean distance between the keypoints is used as a distance metric and the metric is calculated on the images from the validation dataset.

We compare the models at the distance thresholds of 100, 200, and 500 pixels. This means that a correct pose prediction needs to be closer to ground truth than this threshold for all of the predicted keypoints.

We choose the threshold values in pixels because we have no other means to measure distance in the images. The usual approach would be to make the threshold to some distance on the lynx's body to reflect a distance in the physical world (e.g. withers to tail base distance). However, the lynxes are angled in various ways and in different positions so this kind of measurement fails.

We plot the results for different thresholds in Figure 37. The results at the given thresholds are also in Table 5 in Appendix B. At the threshold 100, the best-performing model is the best HRNet model with 75.41% of correct predictions. The best ResNet-50 model performance is similar with 75.32%. However, ResNet-152 has 72.86% of correct predictions, which means the trained model has more difficulty with fine predictions than the other best models. The best models achieve more than 45% of correct predictions compared to the baseline models and for HRNet and ResNet-50 roughly 5% more than the models trained without augmentations.

a similar trend can be observed at the threshold value of 200 pixels. HRNet is the best-performing model with 91.48% of correct predictions at this threshold. ResNet-50 is again performing comparably with 91.25%. ResNet-152 manages to decrease the performance difference observed at the previous threshold to roughly 1% with 90.22%. The percentages of the best models again outperform the baseline models by roughly 30% to 45% and the models trained without augmentations by roughly 5%.

At the last threshold value of 500, HRNet is again the best-performing model with 98.77%

of correct predictions. At this threshold, both the ResNet-50 with 98.34% and the ResNet-152 with 98.1% show comparable results to HRNet. The difference in the performance of the best models and the baseline models ranges roughly from 12% to 21%. The difference between the best models and models trained without augmentations is around 1%.

The augmentations helped to increase the percentage of correct predictions tremendously over the baseline models. The percentage increase over the models without the augmentations is not as big, but the values achieved at threshold 100 show that the augmentations help to achieve finer predictions.

7.3.3 Visualization

Having presented the experiment results, we now compare the model pose output to the ground truth pose. First, we compare only the best models' outputs to the ground truth. These models are the best HRNet, ResNet-50, and ResNet-152 models from Table 4. All of the images we present here were chosen from the validation dataset, meaning the models' outputs are done on unseen data. They were also zoomed to show the area close to their ground truth bounding box to better visualize the pose estimation results.

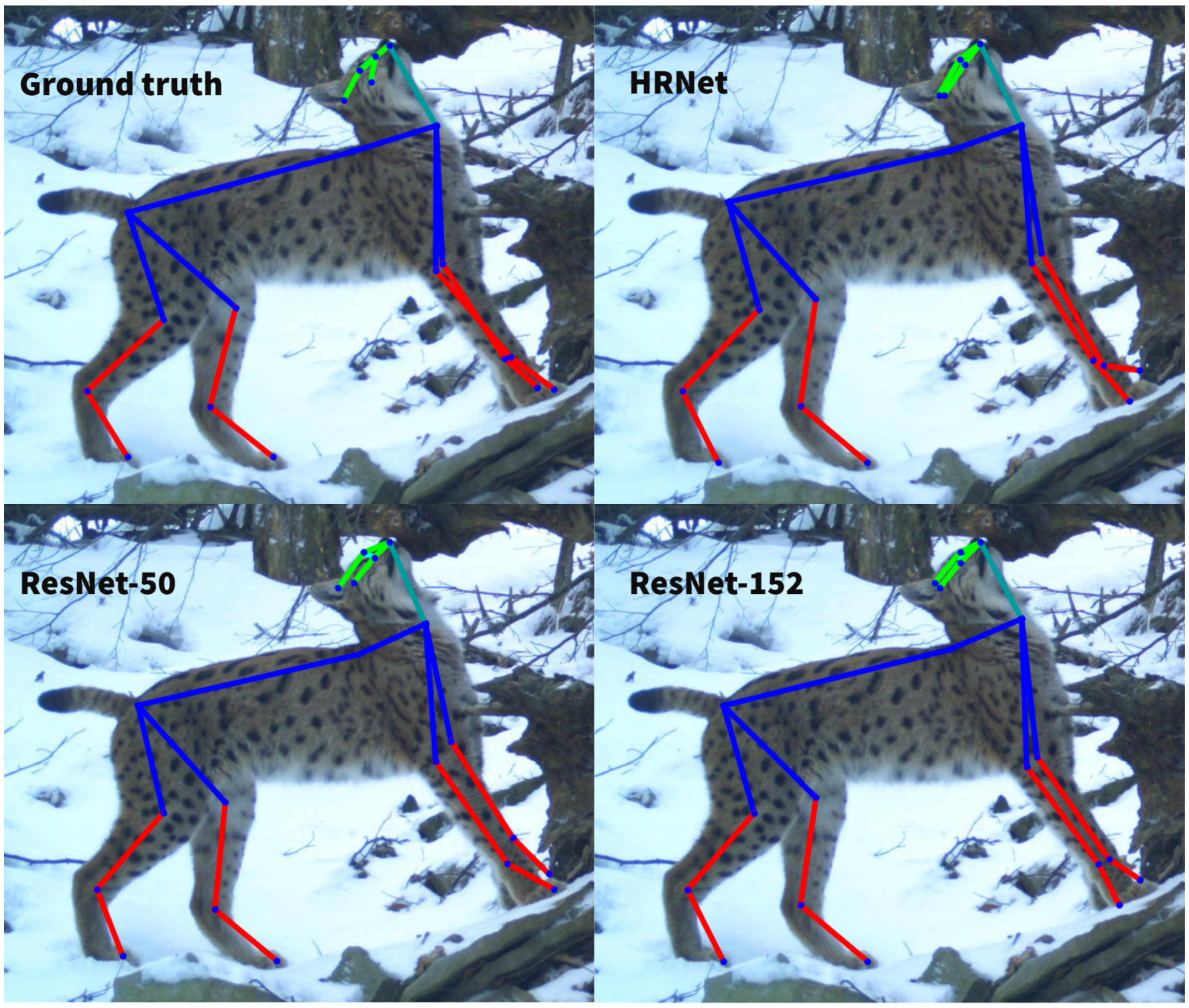


Figure 38: The ground truth pose and the poses predicted by the best models from Table 4 on an image from the validation dataset.

We show the models' predicted poses in Figures 38 and 39. In Figure 38, we see the lynx in a difficult pose, where its left front paw is almost entirely occluded by the right front paw and the head is tilted back in an unusual position. We notice that the models' predictions for the back legs, tail base, and withers are very similar and close to ground truth. The occluded front paw caused an issue for all of the models, but we also observe lower confidence for these keypoints - the networks output confidence around 0.5 as opposed to the non-occluded paw which has a confidence score around 0.7. The same occlusion issue applies to the head keypoints. The models did manage to predict the pose for the unusual head position as the nose is almost perfectly placed for all of the models. The confidence scores reflect this by being around 0.9. The eye keypoints placement is hard since the left eye is occluded and even the ground truth annotation could be up for discussion. However, the occlusion is not reflected in the confidence score of the models. The models have higher confidence in the occluded left eye keypoint placement (HRNet has 0.81 confidence for the left eye as opposed to 0.72 for the right eye). We notice that the models try to connect the nose, eye, and ear base keypoints with a straight line. This behavior is according to the expectation we discussed in Section 2.1.

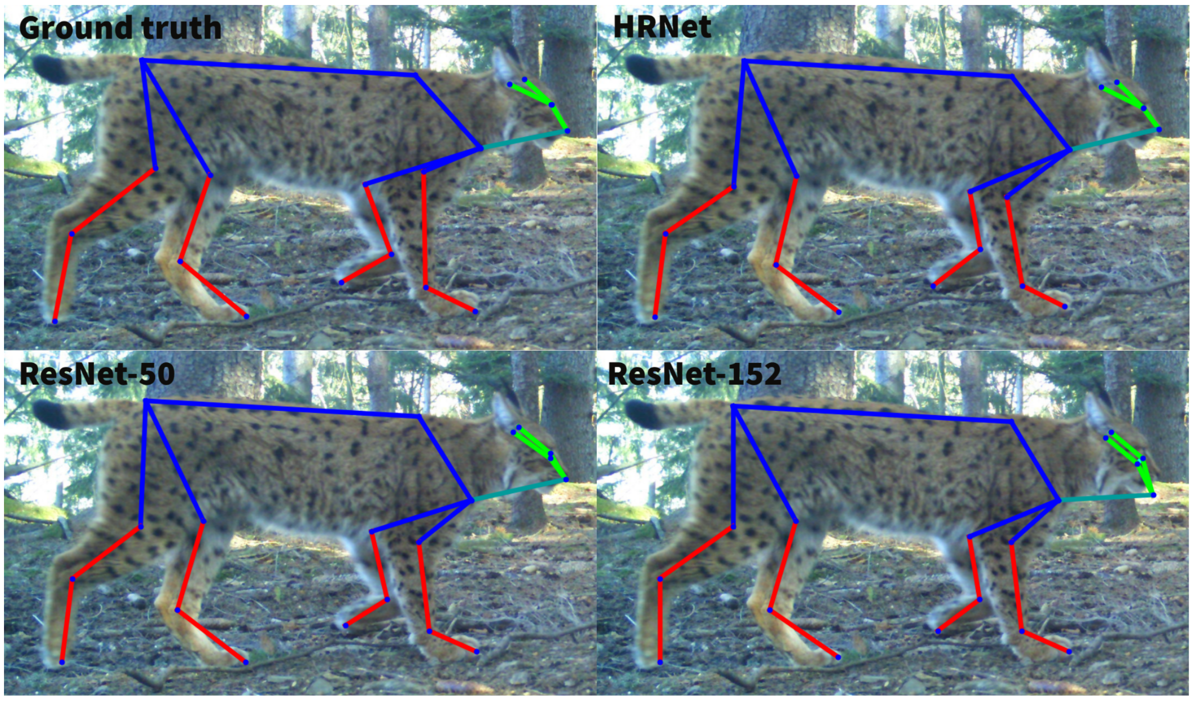


Figure 39: The ground truth pose and the poses predicted by the best models from Table 4 on an image from the validation dataset.

In Figure 39, we see the lynx in an almost perfect position for pose estimation. This is because the lynx is visible clearly and the only occluded keypoints are the left eye and the left ear base. However, the head position in the image makes these keypoints easily predictable. The HRNet prediction and the ground truth position of these keypoints correspond to each other closely. ResNet-50 model predicted the occluded keypoints slightly below the ground truth annotation. ResNet-152 failed to place the head keypoints and confused the nose with the sun patch in the background. The model probably tried to keep the distance between the eyes and the nose relatively close. Because of this, it fixated the eye keypoints onto the black patches below the actual eyes. This failure is at least somewhat reflected in the confidence

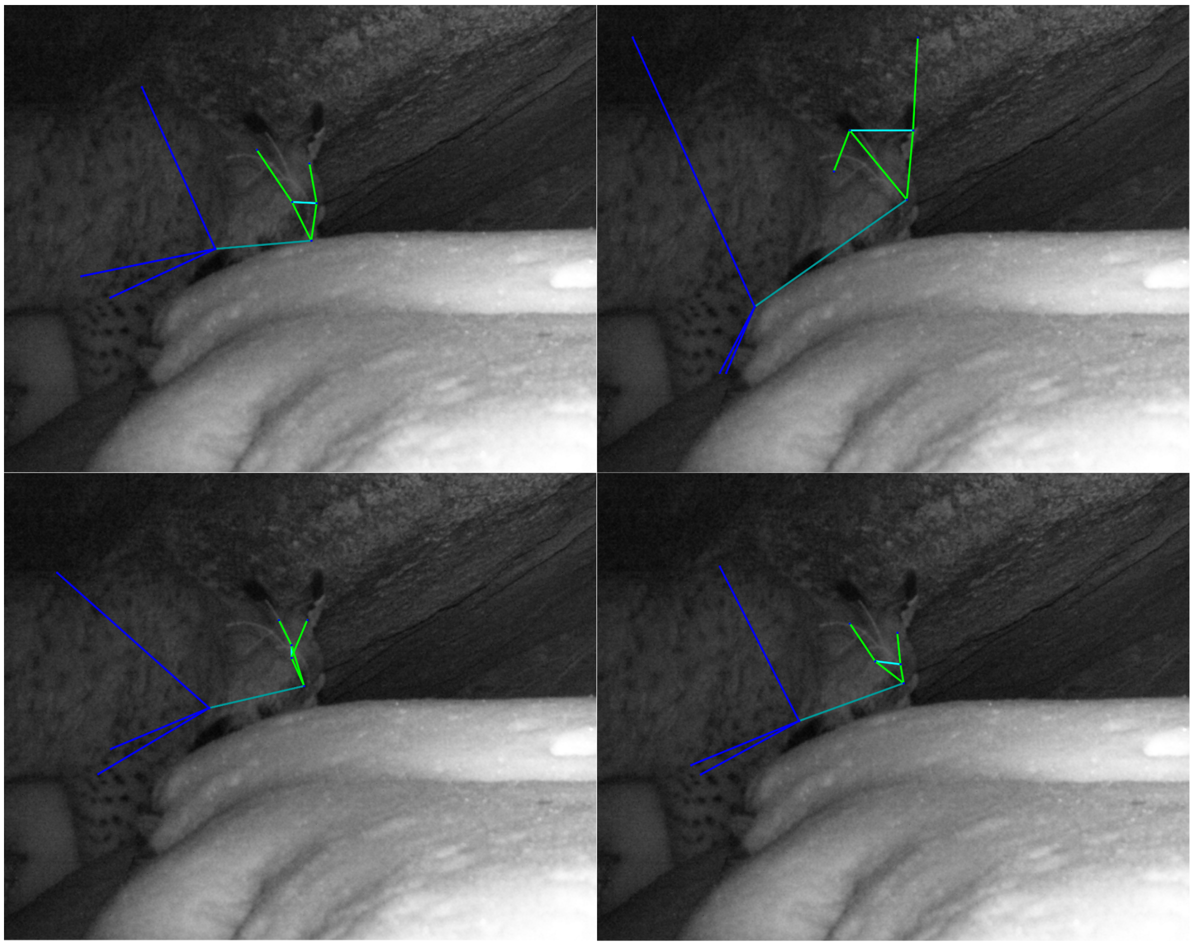


Figure 40: HRNet predictions of different models. On the chosen image, the baseline HRNet model presented in Table 1 had the worst OKS value out of all validation images. Top left image is ground truth pose, top right image is the baseline model, bottom left image is the model trained without augmentations from Section 7.2, bottom right is the best HRNet model presented in Table 4 trained in the color or noise experiment.

scores - the nose, which was misplaced completely has a confidence score of 0.73 (compared to 0.83 for ResNet-50 and 0.89 for HRNet). The rest of the keypoints were placed almost perfectly by the models. The one visible difference between the models and the ground truth is in the right back knee keypoint placement. The models all place the keypoint in a similar location below the actual position. The confidence scores are also relatively high for the models (around 0.86). This could mean that there are annotations in the training ground truth data which are placing this keypoint lower than it actually is. However, we have not observed this behavior in the previous Figure.

Having tested the models on handpicked images for the validation dataset, we now take the images for which the baseline models presented in Section 7.2 have the worst OKS values. We take the ground truth pose and compare it to three models - the baseline model, the model fine-tuned using no augmentations, and the best model. We do this comparison for all of the architectures and we visualize only the keypoints annotated in the ground truth data.

We always visualize the ground truth on the top left image, the baseline model on the

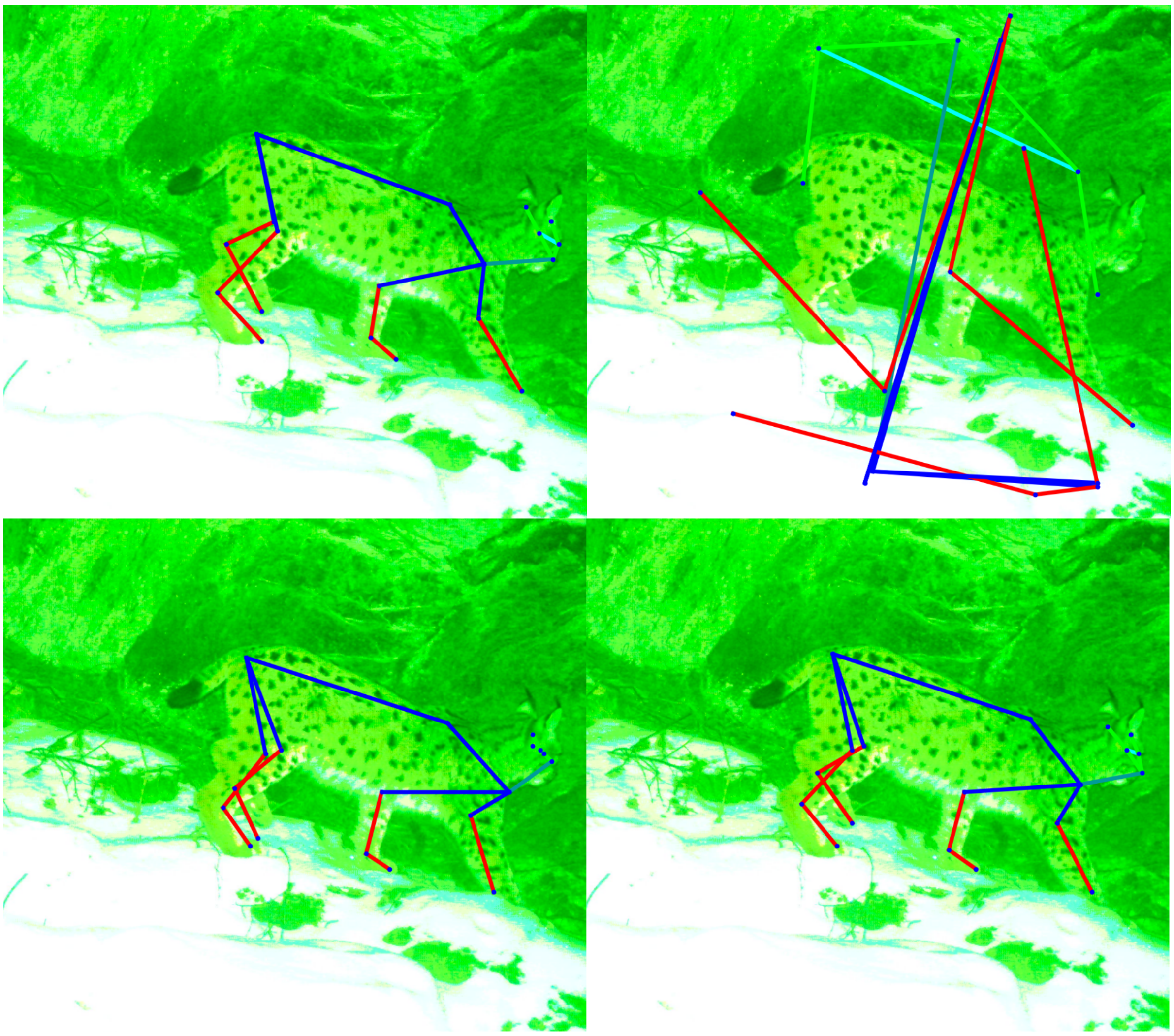


Figure 41: ResNet-50 predictions of different models. On the chosen image, the baseline ResNet-50 model presented in Table 1 had the worst OKS value out of all validation images. Top left image is ground truth pose, top right image is the baseline model, bottom left image is the model trained without augmentations from Section 7.2, bottom right is the best ResNet-50 model presented in Table 4 trained in the color or noise experiment.

top right image, the model using no augmentations on the bottom left, and the best model on the bottom right.

In Figure 40 we visualize the image which has the worst OKS value for the baseline HRNet model. There are only 9 visible keypoints in the image, the rest is occluded or out of image bounds. We notice that most of the keypoints predicted by the baseline model are not even located on the lynx. The aforementioned line that should form between the nose, eyes, and ears is present at all and the lynx's body proportions are also wrong. The model trained without any augmentations is better with its throat and the front knees predictions. The withers prediction is at least located on the lynx's back. The head keypoints are also located on the lynx as opposed to the baseline model, but the pose is very deformed. The ear keypoints are close to the ground truth positions. The best model made an error only in the lynxes head, and the predicted head keypoints at least somewhat correspond to the

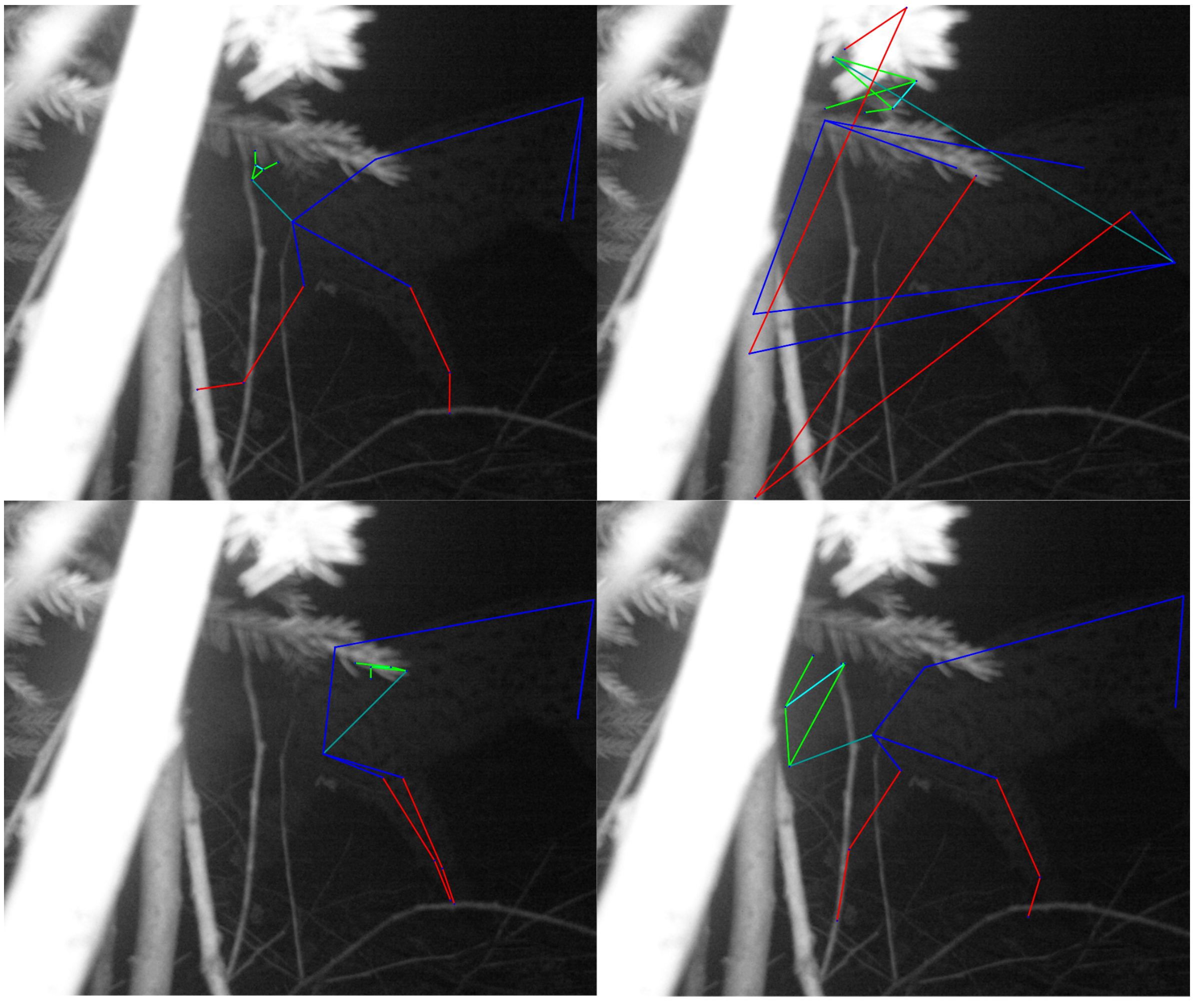


Figure 42: ResNet-152 predictions of different models. On the chosen image, the baseline ResNet-152 model presented in Table 1 had the worst OKS value out of all validation images. Top left image is ground truth pose, top right image is the baseline model, bottom left image is the model trained without augmentations from Section 7.2, bottom right is the best ResNet-152 model presented in Table 4 trained in the noise then color experiment.

ground truth. However, the nose and eyes keypoint are located higher than in the ground truth. The throat and withers keypoints are located almost perfectly as well as the right knee. The left knee is occluded in the image and maybe should not have been annotated as we agree more with the model's prediction than with the ground truth placement.

In Figure 41 we show the image with the worst OKS value for the baseline ResNet-50 model. The lynx in this image is almost entirely visible and the difficulty comes from the green image color and the camera noise that is present in the image. The keypoints in this image are all visible except the right front paw. The baseline model is not able to handle the green image at all with the prediction being all over the image and outside of the ground truth bounding box (the box is not visualized in the image). The model's performance improves drastically for the model trained without augmentations and the predicted pose resembles the actual pose in most of the keypoints. It misplaced the left back leg and the head keypoints are deformed as in the HRNet model without augmentations. The other keypoints are positioned sufficiently. The best model handles the green color and the image noise with ease, which could be because of the color and noise augmentations usage. The left back leg prediction is better than the previous model's prediction. The head keypoints are again a little off with the left ear being slightly below the ground truth. The same applies to the occluded left eye, which should be slightly more to the right. However, we can see the deformation of the head from the previous model being fixed as the predicted head keypoints resemble the ground truth head keypoints.

In Figure 42 we depict the image with the worst OKS value for the baseline ResNet-152. This image is hard for pose estimation, because of the occlusion and the low image brightness. The ground truth for this image is also poorly annotated with the head keypoints being out of proportion with no head visible in the image. The right front paw is occluded too and the annotation for it also seems out of proportion compared to the other paw. The withers keypoint should probably be located closer to the lynx's head as well.

The baseline model once again fails to predict the pose with most of the keypoints being out of the bounding box (the box is not visualized in the image). The model trained without any augmentations also fails to predict the pose but manages to locate withers, the tail base, and the left front leg. The head keypoints are deformed again and fixated on the twig in the foreground. The best model managed to do a very good job with the pose prediction. The predicted head keypoints also make the head look deformed in a way, but it is much more in proportion to the body than the ground truth one. Interestingly the left ear is deformed and placed in almost the same location as the left eye keypoint. The head keypoints placement also makes more sense with regard to the rest of the body. The withers keypoint is located where it should have been annotated in the ground truth. The left back knee is placed to the exact same coordinates as the right back knee and should be more in front. The length of the right front paw is also out of proportion to the left front paw and since it is also occluded, we can not say if the ground truth or the ResNet-152 placement is better.

Having visualized the predictions on the images with the worst OKS values for the models, we notice that the final pose models were able to adapt and predict the poses of the lynxes. Only the HRNet prediction of the head keypoints was insufficient, but we saw this not being an issue for the other images where HRNet visually achieved a better pose prediction than the other models. The augmentations helped the models to predict the pose better as the mistakes done by the models trained without augmentations were corrected by the best models. We also noticed head keypoints deformations for all of the models trained without the augmentations. This could be caused by the models overfitting to training data and being unable to place the head keypoints correctly.

We also visualize the pose estimation progression throughout the epochs in Figure 44 in Appendix C. The visualization is done on an image from the validation dataset for which the baseline HRNet model has the second-worst OKS value. The experiment visualized is the one color or noise experiment where the best HRNet model was trained. We notice the predicted pose gradually improving as the model learns. The biggest difference is between the baseline model and the model in epoch 10. We notice that the model is able to quickly adapt to the blurry lynx and the differences between the subsequent epochs are more subtle. An interesting phenomenon is the pose predicted in the 40th epoch. This estimated pose is almost the same as the ground truth pose and there is a big decrease in the quality of the predicted pose in epoch 60 compared to the 40th epoch. However, the overall mAP value on the validation dataset for epoch 60 (0.9025 mAP) is better than for epoch 40 (0.8847 mAP).

7.4 Importance of initialization

So far we have experimented with the heatmap-based pose models. In this Section, we try to train the pose model with a keypoint head and try different initialization weights for the model to show the pre-training importance.

For this experiment, we choose HRNet as our backbone architecture. We replace the heatmap head with the DeepPose regression head introduced in [66] and implemented in MMPose [51]. To connect the outputs from the convolutional layers into the fully connected layers, we use global average pooling. For the training, we use the smooth-L_1 loss from Section 3.3 for all runs except one where we try to run an experiment with the MSE loss from the same Section. The weights of the keypoint head are always randomly initialized without any weights transferred. This means that the models start with a zero mAP value on the validation dataset as we will see later.

The models are trained for 250 epochs with 5 epochs at the start being a linear warmup with the warmup ratio of $1e^{-4}$. We use Adam optimizer with the initial learning rate (after warmup) of $5e^{-4}$ and step scheduler with a learning rate decay factor of 0.5. We use 6 steps at epochs 10, 60, 130, 180, 219, and 220.

We depict the results of the experiment runs in Figure 43. The runs which were trained with the Animal-Pose backbone are denoted as "Animal-Pose initialization", here we experiment with the two losses and find no real difference in the overall mAP value progression and the final mAP values. Hence we use smooth-L_1 loss for the rest of the runs. The runs trained with the backbone finetuned on the Lynx-Pose dataset are denoted as "Lynx-Pose initialization". This backbone is taken from the HRNet trained on the Color or Noise experiment presented in Section 7.3.2. The last three runs are denoted as "Random initialization" and the weights for the whole model are chosen randomly at the initialization.

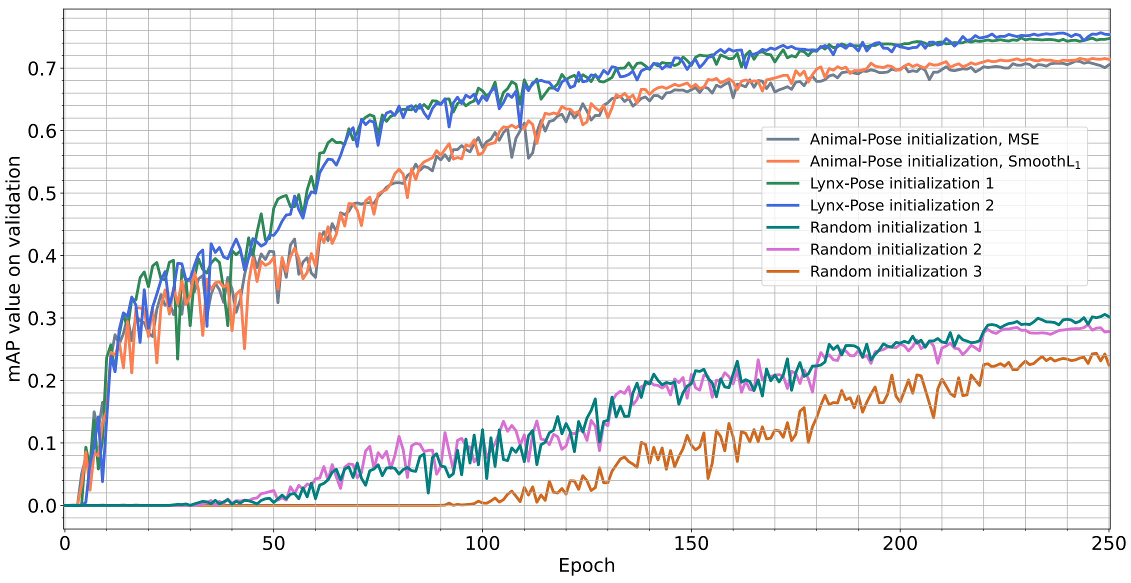


Figure 43: The comparison of validation mAP progression in the experiments with the different backbone initializations.

In Figure 43, we see the ascent of the mAP value achieved by the Animal-Pose starts slowing down around epoch 200. The final values reached by these models are placed around 0.71. Similar behavior can be observed in the results achieved by the Lynx-Pose models.

We notice a slower ascent after the 200th epoch with a final mAP value around 0.75. The mAP value progression for models with random initialization starts slowing down after the last scheduler steps at epochs 219 and 220 and stays around the mAP value of 0.28.

We notice the models could have been trained to a better final precision if we did not use the last two scheduler steps at once and trained the networks for a longer time. This is especially noticeable for the randomly initialized models. However, reaching the best possible values is not the main goal of this experiment. We want to show the importance and the effect of pre-trained weights on the training.

We observe that the pre-trained weights help tremendously with the initial and the final performance. Both of the pre-trained models' performances start rising quickly and reach better mAP values. It is obviously not known if the randomly initialized networks would show better results than the pre-trained networks if we let them run longer. The pre-trained models achieve these "good" mAP values in a reasonable amount of time. We observe that in the initial training stages, the pre-trained models show a similar increase in performance. The interesting thing is the simultaneous rise in the mAP value of the Lynx-Pose models in epoch 50. This is the point where these models overtake the Animal-Pose models in performance and it stays this way until the end of the experiment.

Another observation we make is the difference between the best keypoint model performance (mAP value of 0.7568) and the best heatmap model performance (mAP value of 0.9180). This is likely caused by the pre-trained heatmap head used for the heatmap model as opposed to the randomly initialized keypoint head. This shows the importance of the pre-trained weight even when it comes to a single network module and it would be an interesting topic for further studies.

8 Conclusion

In my thesis, the animal pose estimation task was tackled. The goal of the work was to train a pose estimation model for the Eurasian lynx using the existing animal pose estimation methods.

First, we introduced the animal pose estimation task and specified it for the lynx pose estimation. We defined the limitations and constraints of our task, mainly with regard to the expected input and output.

Then we described the two main deep neural network architectures used as a backbone for the task of top-down 2D animal pose estimation, ResNet and HRNet. Along with the architectures, we also studied the current approaches to the neural network output - the keypoint and the heatmap regression approaches. We discussed the advantages and shortcomings of the approaches. We also briefly introduced MSE loss and smooth-L_1 loss commonly used for training.

Afterward, we studied the methods for pose estimation evaluation. We focused mainly on the object keypoint similarity metric and the different interpretations of the scale term used to calculate it. We proposed a solution to solve the numerical issues which emerged with one scale term interpretation. Then we presented how object keypoint similarity is used to calculate the commonly used mean average precision for pose estimation. We also studied two other metrics for pose estimation evaluation - the percentage of correct keypoints and the percentage of correct predictions.

We also presented the datasets available for the animal pose estimation tasks. First, we introduced our novel Lynx-pose dataset for which we want to solve the pose estimation task. Then, we introduced other datasets with the main focus on pose estimation for quadruped mammals. The best dataset for our task was the Animal-Pose dataset which estimates pose for a very similar set of keypoints for which pre-trained models exist in the MMPose toolbox.

In the last theoretical part of my thesis, image augmentations were introduced. We have chosen augmentations that are all available in the MMPose and Albumentations toolboxes to be able to later use them in our experiments. We divided the augmentations into five groups and presented their input parameters and their modus operandi.

In the practical part of my thesis, the models for lynx pose estimation were trained. First, we introduced the models we chose to train and the settings of the experiment. We trained three models - HRNet-W32, ResNet-50, and ResNet-152 all pre-trained on the aforementioned Animal-Pose dataset. We used the heatmap approach to pose estimation to fully utilize the pre-training of the models. All of the models were implemented in the MMPose toolbox and the training was also conducted with the same toolbox. After introducing the models and settings, we used the presented augmentation techniques to augment the training images during training. Using these augmentations, we conducted several experiments to find the best model. We used the presented mean average precision measured on the validation dataset as the main evaluation metric. After finding the best models, we validate their performance with the use of the percentage of correct predictions metric.

During the experiments, we observed that HRNet-W32 is consistently the best-performing model. It achieved the best mean average precision value in most of the experiments and also the best overall mean average precision value. This statement was also true for the percentage of correct predictions, where it also outperformed the other models and had the best ability to conduct fine-grained pose estimation. We also observed that while the best ResNet-152 model has a better mean average precision value than the best ResNet-50 model, it is worse than ResNet-50 regarding the percentage of correct predictions, especially at a low threshold. This can also somewhat be seen in the visualization of the model outputs. We observed that the trained ResNet-152 model had a hard time estimating the lynx's head pose for a supposedly "easy" image.

In the last part of the thesis, we experiment with the direct keypoint regression approach and also show the importance of pre-training for the network. We showed that without the pre-trained weights, the model is not able to reach a sufficient mean average precision value and would require significantly more epochs to train. We also observe the importance of the pre-trained heatmap regression head. We used a randomly initialized keypoint head in the experiments and the mean average precision value for the best model had mAP value lower by 0.15 than the best heatmap model.

We provide all the source files and the weights for the trained models on `https://github.com/Tmajer/DP-Carnivore`. The functionality of the pose models can also be tested on `https://huggingface.co/spaces/tmajer/Lynx-Pose-Estimation`.

8.1 Future Work

Having successfully developed and trained a model for lynx pose estimation, we dedicate this chapter to the introduction of the new questions and challenges that have arisen from our empirical findings. We introduce the areas for future research and development in lynx pose estimation that emerged in the experimental part of the thesis.

The first possible area of research is the augmentations. Since we experimented with only a few of the possible combinations of augmentations, more studies on this topic could further increase the performance of the models. Since we made several observations based on the training runs, we also lay a foundation for a statistical study of the effect of augmentations. The other hyperparameters used in the training pipeline were also fixed for our experiments and different settings that would allow for faster or better training could be found.

The second area of research we found is a multi-head approach to pose estimation. It would be interesting to experiment with a combination of a direct keypoint regression head and a heatmap regression head. There are several tasks that arise ranging from the implementation and integration of such head into the MMPose ecosystem to finding a good aggregation function to combine the outputs of the heads.

References

[1] J. J. Audubon. *Ornithological biography*, volume 1. 1832.

[2] J. F. Banzi. A sensor based anti-poaching system in tanzania national parks. *International Journal of Scientific and Research Publications*, 4(4):1–7, 2014.

[3] V. Belagiannis and A. Zisserman. Recurrent human pose estimation. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 468–475. IEEE, 2017.

[4] B. Biggs, O. Boyne, J. Charles, A. Fitzgibbon, and R. Cipolla. Who left the dogs out? 3d animal reconstruction with expectation maximization in the loop. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 195–211. Springer, 2020.

[5] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin. Albumentations: fast and flexible image augmentations. *Information*, 11(2):125, 2020.

[6] J. Cao, H. Tang, H.-S. Fang, X. Shen, C. Lu, and Y.-W. Tai. Cross-domain adaptation for animal pose estimation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[7] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):172–186, 2021.

[8] L. Catarinucci, R. Colella, L. Mainetti, L. Patrono, S. Pieretti, A. Secco, and I. Sergi. An animal tracking system for behavior analysis using radio frequency identification. *Lab animal*, 43(9):321–327, 2014.

[9] Y. Chen, Z. Wang, Y. Peng, Z. Zhang, G. Yu, and J. Sun. Cascaded pyramid network for multi-person pose estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7103–7112, 2018.

[10] P. E. Clark, D. E. Johnson, M. A. Kniep, P. Jermann, B. Huttash, A. Wood, M. Johnson, C. McGillivan, and K. Titus. An advanced, low-cost, gps-based animal tracking system. *Rangeland Ecology & Management*, 59(3):334–340, 2006.

[11] M. Contributors. Openmmlab's pre-training toolbox and benchmark. `https://github.com/open-mmlab/mmpretrain`, 2023.

[12] T. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models-their training and application. *Comput. Vis. Image Underst.*, 61:38–59, 1995.

[13] J. Dorfling, S. B. Siewert, S. Bruder, C. Aranzazu-Suescun, K. Rocha, P. D. Landon, G. Bondar, T. Pederson, C. Le, R. Mangar, et al. Satellite, aerial, and ground sensor fusion experiment for management of elephants and rhinos and poaching prevention. In *AIAA SCITECH 2022 Forum*, page 1270, 2022.

[14] A. M. Dujon, R. T. Lindstrom, and G. C. Hays. The accuracy of fastloc-gps locations and implications for animal tracking. *Methods in Ecology and Evolution*, 5(11):1162–1169, 2014.

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html.

[16] R. E. Floyd. Rfid in animal-tracking applications. *IEEE Potentials*, 34(5):32–33, 2015.

[17] D. Fourure, R. Emonet, E. Fromont, D. Muselet, A. Tremeau, and C. Wolf. Residual conv-deconv grid network for semantic segmentation. In *BMVC 2017*, 2017.

[18] F. A. Francisco, P. Nührenberg, and A. Jordan. High-resolution, non-invasive animal tracking and reconstruction of local environment in aquatic ecosystems. *Movement ecology*, 8(1):1–12, 2020.

[19] W. J. Gibbons and K. M. Andrews. Pit tagging: simple technology at its best. *Bioscience*, 54(5):447–454, 2004.

[20] O. Gimenez, S. Gatti, C. Duchamp, E. Germain, A. Laurent, F. Zimmermann, and E. Marboutin. Spatial density estimates of eurasian lynx (lynx lynx) in the french jura and vosges mountains. *Ecology and Evolution*, 9(20):11707–11715, 2019.

[21] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[22] J. M. Graving, D. Chae, H. Naik, L. Li, B. Koger, B. R. Costelloe, and I. D. Couzin. Deepposekit, a software toolkit for fast and robust animal pose estimation using deep learning. *eLife*, 8:e47994, oct 2019.

[23] C. He, Y. Kong, Y. Liu, C. Ma, T. Li, Q. Li, S. Li, and D. Wang. Determining the daily activity pattern of chinese mountain cat (felis bieti): A comparative study based on camera-trapping and satellite collar tracking data. *Biodiversity Science*, 30(9):22081, 2022.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] M. Hebblewhite and D. T. Haydon. Distinguishing technology from biology: a critical review of the use of gps telemetry data in ecology. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1550):2303–2312, 2010.

[26] T. Hodaň. *Pose Estimation of Specific Rigid Objects.* PhD thesis, Czech Technical University, 2021.

[27] G. Huang, D. Chen, T. Li, F. Wu, L. Van Der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv preprint arXiv:1703.09844*, 2(2), 2017.

[28] V. R. Jain, R. Bagree, A. Kumar, and P. Ranjan. wildcense: Gps based animal tracking system. In *2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 617–622. IEEE, 2008.

[29] L. Jiang, C. Lee, D. Teotia, and S. Ostadabbas. Animal pose estimation: A closer look at the state-of-the-art, existing gaps and opportunities. *Computer Vision and Image Understanding*, page 103483, 2022.

[30] S. X. Ju, M. J. Black, and Y. Yacoob. Cardboard people: a parameterized model of articulated image motion. *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, pages 38–44, 1996.

[31] J. Kamminga, E. Ayele, N. Meratnia, and P. Havinga. Poaching detection technologies—a survey. *Sensors*, 18(5):1474, 2018.

[32] R. Kays, M. C. Crofoot, W. Jetz, and M. Wikelski. Terrestrial animal tracking as an eye on life and planet. *Science*, 348(6240):aaa2478, 2015.

[33] R. Kays, S. Tilak, M. Crofoot, T. Fountain, D. Obando, A. Ortega, F. Kuemmeth, J. Mandel, G. Swenson, T. Lambert, et al. Tracking animal location and activity with an automated radio telemetry system in a tropical rainforest. *The Computer Journal*, 54(12):1931–1948, 2011.

[34] M. J. Kelly, J. Betsch, C. Wultsch, B. Mesa, and L. S. Mills. Noninvasive sampling for carnivores. *Carnivore ecology and conservation: a handbook of techniques*, pages 47–69, 2012.

[35] S. Krishnamurthy and S. Gayathri. Prevention of poaching of tigers using wireless sensor network. In *2017 IEEE International Conference on Antenna Innovations & Modern Technologies for Ground, Aircraft and Satellite Applications (iAIM)*, pages 1–7. IEEE, 2017.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[37] R. Labuguen, J. Matsumoto, S. B. Negrete, H. Nishimaru, H. Nishijo, M. Takada, Y. Go, K.-i. Inoue, and T. Shibata. Macaquepose: A novel "in the wild" macaque monkey pose dataset for markerless motion capture. *Frontiers in behavioral neuroscience*, 14:581154, 2021.

[38] C. Li and G. H. Lee. Coarse-to-fine animal pose and shape estimation. *Advances in Neural Information Processing Systems*, 34:11757–11768, 2021.

[39] J. Li, S. Bian, A. Zeng, C. Wang, B. Pang, W. Liu, and C. Lu. Human pose regression with residual log-likelihood estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11025–11034, 2021.

[40] S. Li, J. Li, H. Tang, R. Qian, and W. Lin. Atrw: A benchmark for amur tiger reidentification in the wild. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2590–2598, 2020.

[41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[42] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[43] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. Smpl: a skinned multi-person linear model. *ACM Trans. Graph.*, 34:248:1–248:16, 2015.

[44] M. Lourakis and X. Zabulis. Model-based pose estimation for rigid objects. In *Computer Vision Systems: 9th International Conference, ICVS 2013, St. Petersburg, Russia, July 16-18, 2013. Proceedings 9*, pages 83–92. Springer, 2013.

[45] Z. Luo, Z. Wang, Y. Huang, L. Wang, T. Tan, and E. Zhou. Rethinking the heatmap regression for bottom-up human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13264–13273, 2021.

[46] M. Marks, Q. Jin, O. Sturman, L. von Ziegler, S. Kollmorgen, W. von der Behrens, V. Mante, J. Bohacek, and M. F. Yanik. Deep-learning-based identification, tracking, pose estimation and behaviour classification of interacting primates and mice in complex environments. *Nature machine intelligence*, 4(4):331–340, 2022.

[47] J. Marstaller, F. Tausch, and S. Stock. Deepbees-building and scaling convolutional neuronal nets for fast and large-scale visual monitoring of bee hives. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[48] A. Mathis, T. Biasi, S. Schneider, M. Yuksekgonul, B. Rogers, M. Bethge, and M. W. Mathis. Pretraining boosts out-of-domain robustness for pose estimation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1859–1868, 2021.

[49] S. Matuska, R. Hudec, P. Kamencay, M. Benco, and M. Radilova. A novel system for non-invasive method of animal tracking and classification in designated area using intelligent camera system. *Radioengineering*, 25(1):161, 2016.

[50] Y. Miao and W. Luo. Improve generalization ability of cnn by data augmentation and se block in landmark classification. In *2022 14th International Conference on Computer Research and Development (ICCRD)*, pages 250–255. IEEE, 2022.

[51] MMPose Contributors. Openmmlab pose estimation toolbox and benchmark. https://github.com/open-mmlab/mmpose, 2020.

[52] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pages 483–499. Springer, 2016.

[53] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[54] T. D. Pereira, D. E. Aldarondo, L. Willmore, M. Kislin, S. S.-H. Wang, M. Murthy, and J. W. Shaevitz. Fast animal pose estimation using deep neural networks. *Nature methods*, 16(1):117–125, 2019.

[55] E. Pesenti and F. Zimmermann. Density estimations of the eurasian lynx (lynx lynx) in the swiss alps. *Journal of Mammalogy*, 94(1):73–81, 2013.

[56] M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[57] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[58] H. Sawyer, M. J. Kauffman, R. M. Nielson, and J. S. Horne. Identifying and prioritizing ungulate migration routes for landscape-level conservation. *Ecological Applications*, 19(8):2016–2025, 2009.

[59] F. Seebacher and E. Post. Climate change impacts on animal migration. *Climate Change Responses*, 2(1):1–2, 2015.

[60] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[61] X. Sun, B. Xiao, F. Wei, S. Liang, and Y. Wei. Integral human pose regression. In *Proceedings of the European conference on computer vision (ECCV)*, pages 529–545, 2018.

[62] M. Sunquist and F. Sunquist. *Wild cats of the world*. University of chicago press, 2017.

[63] R. Takahashi, T. Matsubara, and K. Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):2917–2931, 2019.

[64] B. Thomas, J. D. Holland, and E. O. Minot. Wildlife tracking technology options and cost considerations. *Wildlife Research*, 38(8):653–663, 2011.

[65] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *Advances in neural information processing systems*, 27, 2014.

[66] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1653–1660, 2014.

[67] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364, 2020.

[68] K. Weingarth, C. Heibl, F. Knauer, F. Zimmermann, L. Bufka, and M. Heurich. First estimation of eurasian lynx (lynx lynx) abundance and density using digital cameras and capture–recapture techniques in a german national park. *Animal biodiversity and conservation*, 35(2):197–207, 2012.

[69] H. J. Williams, L. A. Taylor, S. Benhamou, A. I. Bijleveld, T. A. Clay, S. de Grissac, U. Demšar, H. M. English, N. Franconi, A. Gómez-Laich, et al. Optimizing the use of biologgers for movement ecology research. *Journal of Animal Ecology*, 89(1):186–206, 2020.

[70] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 466–481, 2018.

[71] Y. Xu, J. Zhang, Q. Zhang, and D. Tao. Vitpose+: Vision transformer foundation model for generic body pose estimation. *arXiv preprint arXiv:2212.04246*, 2022.

[72] H. Yu, Y. Xu, J. Zhang, W. Zhao, Z. Guan, and D. Tao. Ap-10k: A benchmark for animal pose estimation in the wild. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[73] F. Zhang, X. Zhu, H. Dai, M. Ye, and C. Zhu. Distribution-aware coordinate representation for human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7093–7102, 2020.

[74] Q. Zhao, T. W. Arnold, J. H. Devries, D. W. Howerter, R. G. Clark, and M. D. Weegman. Land-use change increases climatic vulnerability of migratory birds: Insights from integrated population modelling. *Journal of Animal Ecology*, 88(10):1625–1637, 2019.

[75] C. Zheng, W. Wu, T. Yang, S. Zhu, C. Chen, R. Liu, J. Shen, N. Kehtarnavaz, and M. Shah. Deep learning-based human pose estimation: A survey. *ArXiv*, abs/2012.13392, 2019.

[76] B. Zoph, E. D. Cubuk, G. Ghiasi, T.-Y. Lin, J. Shlens, and Q. V. Le. Learning data augmentation strategies for object detection. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII 16*, pages 566–583. Springer, 2020.

[77] S. Zuffi, A. Kanazawa, D. W. Jacobs, and M. J. Black. 3d menagerie: Modeling the 3d shape and pose of animals. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6365–6373, 2017.

Appendix A - Augmentation parameters

Geometric transformations

Random scale	$s \in \langle 0.5, 1.5 \rangle$
Random rotation	$\theta \in \langle -80°, 80° \rangle$
Random bounding box shift	$s_h \in \langle -0.16, 0.16 \rangle$ $s_v \in \langle -0.16, 0.16 \rangle$
Half-body transformation	$t = 8$

Blur transformations

Averaging blur	$k \in \langle 17, 33 \rangle$
Gaussian blur	$k \in \langle 21, 45 \rangle$
Motion blur	$k \in \langle 23, 49 \rangle$
Median blur	$k \in \langle 29, 57 \rangle$

Color transformations

Channel dropout	$v = 102$
Color jitter	$k_b \in \langle 0.35, 1.15 \rangle$ $k_c \in \langle 0.35, 3.1 \rangle$ $k_s \in \langle 0, 2.6 \rangle$ $k_h \in \langle -0.17, 0.07 \rangle$
HSV transformation	$k_h \in \langle -40, 40 \rangle$ $k_s \in \langle -235, 180 \rangle$ $k_v \in \langle -80, 120 \rangle$
RGB shift	$k_r \in \langle 0, 95 \rangle$ $k_g \in \langle -95, 95 \rangle$ $k_b \in \langle -95, 0 \rangle$
Random brightness and contrast	$k_b \in \langle 0.4, 1.5 \rangle$ $k_c \in \langle -0.8, 0.5 \rangle$

Noise transformations

Downscale	interpolation: nearest neighbor $s \in \langle 0.075, 0.15 \rangle$
Additive Gaussian noise	same for all channels: False $\sigma^2 \in \langle 30, 100 \rangle$ $\mu = 30$
ISO noise	$i \in \langle 0.8, 1 \rangle$ $c \in \langle 0.01, 0.1 \rangle$
Multiplicative noise	scalar/matrix: matrix same for all channels: True $k_m \in \langle 0.45, 1.55 \rangle$
Posterize	$k_r \in \langle 4, 6 \rangle$ $k_g \in \langle 3, 5 \rangle$ $k_b \in \langle 2, 4 \rangle$
Sharpen	$\alpha \in \langle 0.6, 1.0 \rangle$ $l \in \langle 0.6, 1.0 \rangle$

Weather transformations

Random snow	$k_b = 5$ $t \in \langle 0.3, 0.5 \rangle$
Random sun flare	$c = \text{rgb}(255, 255, 255)$ $n \in \langle 6, 8 \rangle$ $r = 100\text{px}$
Random rain	$\theta \in \langle -20°, 20° \rangle$ $k = 9$ $k_b = 0.75$ $c = \text{rgb}(200, 200, 200)$ rain-type: drizzle
Random fog	$k_f \in \langle 0.4, 0.7 \rangle$ $\alpha = 0.05$

Tuned color and noise transformations

Downscale	interpolation: nearest n. $s \in \langle 0.11, 0.18 \rangle$
Additive Gaussian noise	same for all ch.: False $\sigma^2 \in \langle 40, 110 \rangle$ $\mu = 30$
ISO noise	$i \in \langle 0.8, 1 \rangle$ $c \in \langle 0.01, 0.12 \rangle$
Multiplicative noise	scalar/matrix: matrix same for all ch.: True $k_m \in \langle 0.4, 1.5 \rangle$
Posterize	$k_r \in \langle 4, 6 \rangle$ $k_g \in \langle 3, 5 \rangle$ $k_b \in \langle 3, 5 \rangle$
Sharpen	$\alpha \in \langle 0.6, 1.0 \rangle$ $l \in \langle 0.6, 1.0 \rangle$

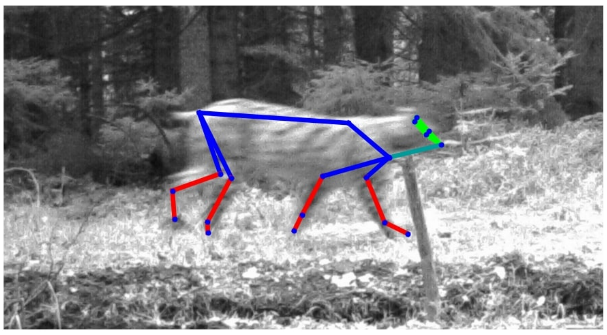
Channel dropout	$v = 102$
Color jitter	$k_b \in \langle 0.425, 1.125 \rangle$ $k_c \in \langle 0.425, 2.325 \rangle$ $k_s \in \langle 0, 2.325 \rangle$ $k_h \in \langle -0.1525, 0.0625 \rangle$
HSV transformation	$k_h \in \langle -42, 42 \rangle$ $k_s \in \langle -145, 145 \rangle$ $k_v \in \langle -75, 75 \rangle$
RGB shift	$k_r \in \langle 0, 105 \rangle$ $k_g \in \langle -105, 105 \rangle$ $k_b \in \langle -105, 0 \rangle$
Random brightness and contrast	$k_b \in \langle 0.675, 1.325 \rangle$ $k_c \in \langle -0.625, 0.625 \rangle$

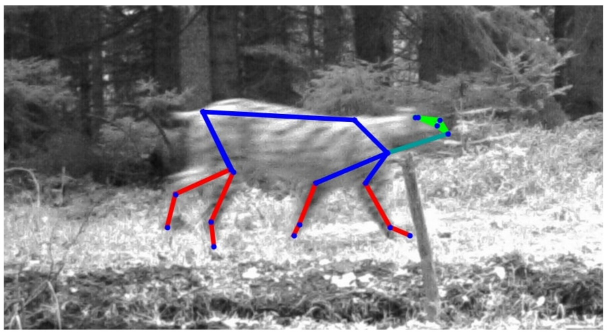
Appendix B - Percentages of Correct Predictions

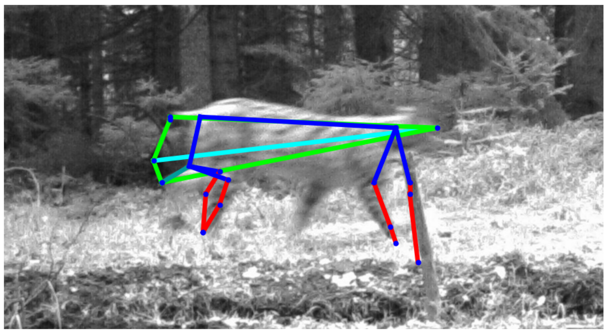
	t=100	t=200	t=500
HRNet Baseline	30.54	61.12	86.36
HRNet No augmentations	69.55	85.94	97.39
HRNet Best	**75.41**	**91.48**	**98.77**
ResNet-50 Baseline	19.67	44.02	77.45
ResNet-50 No augmentations	70.23	87.26	96.45
ResNet-50 Best	75.32	91.25	98.34
ResNet-152 Baseline	24.52	53.21	82.57
ResNet-152 No augmentations	70.49	87.17	97.39
ResNet-152 Best	72.86	90.22	98.10

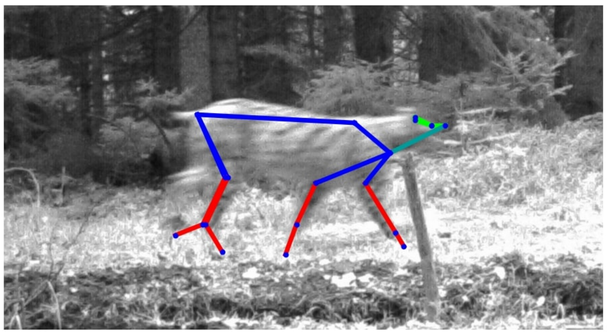
Table 5: The percentages of correct predictions of the models on given thresholds.

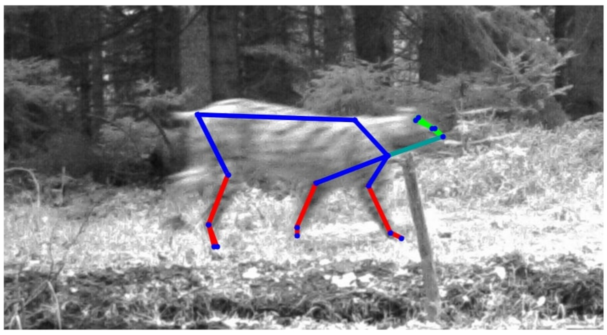
Appendix C - Pose Throughout the Epochs

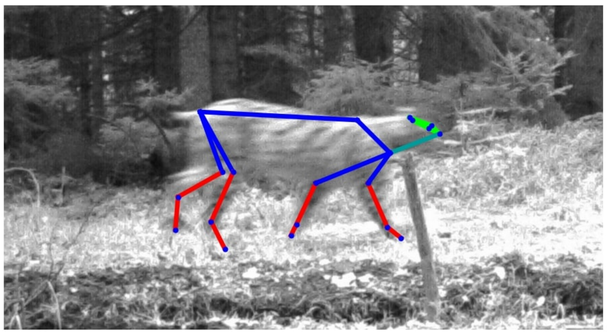

(a) Ground truth pose.

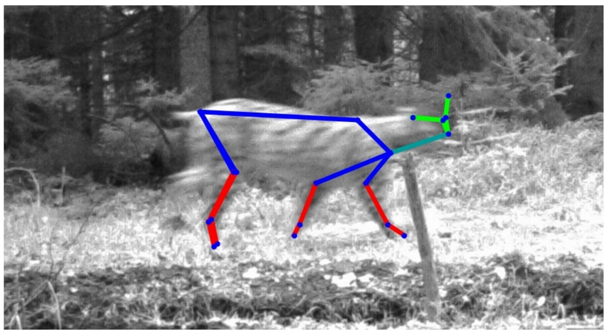

(b) Best model (Epoch 172).


(c) Epoch 0.


(d) Epoch 10.


(e) Epoch 30.


(f) Epoch 40.


(g) Epoch 60.

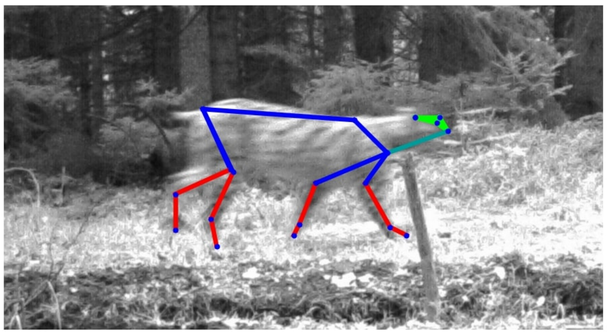

(h) Epoch 140.

Figure 44: The pose estimation progression of HRNet trained during the color or noise augmentations experiment throughout the epochs.