Multi-task Learning For Joint Action and Gesture Recognition

Konstantinos Spathis[1] Nikolaos Kardaris[2] Petros Maragos[1,2,3]

[1]Robotics Institute, Athena Research Center, 15125 Maroussi, Greece
[2]School of Electrical & Computer Engineering, National Technical University of Athens, Greece
[3]HERON - Hellenic Robotics Center of Excellence, Athens, Greece

k.spathis@athenarc.gr, nkardaris@mail.ntua.gr, petros.maragos@athenarc.gr

Abstract

Real-world applications focused on interpreting human behavior often require multiple computer vision tasks to be addressed simultaneously. Multitask learning typically achieves this by jointly training a single deep neural network to learn shared representations, providing efficiency and improving generalization. Although action and gesture recognition are closely related tasks, since they focus on body and hand movements, current state-of-the-art methods handle them separately. In this paper, we show that employing a multi-task learning paradigm for action and gesture recognition results in more efficient, robust and generalizable visual representations, by leveraging the synergies between these tasks. Experiments on multiple action and gesture datasets demonstrate that handling actions and gestures in a single architecture can achieve better performance for both tasks in comparison to their single-task learning variants.

1. Introduction

Understanding human behavior plays a central role in various domains, such as human-robot interaction (HRI), virtual reality, and multimedia content analysis. Human behavior involves a wide range of visual cues, including whole-body actions and hand gestures, all of which may need to be considered at the same time. However, most computer vision systems are specialized to address only a narrow problem. Current machine learning approaches typically achieve this by following a Single-Task Learning (STL) paradigm, where a single deep learning model or an ensemble of models is trained and deployed to perform the desired tasks. This approach overlooks potential commonalities among them that can lead to better generalization through inductive transfer. An emerging method to address multiple tasks at the same time is Multi-Task Learning (MTL) [3], in which a single deep learning architecture is trained across different tasks.

Specifically, MTL aims to improve the performance of a neural network for multiple related tasks by exploiting useful similarities and differences between them. A task refers to a distinct learning problem, which has its own objective function and corresponds to a specific dataset. Tasks can differ depending on the type of learning problem addressed or the differences in the data used, such as variations in camera position and changes in illumination. Therefore different datasets can potentially define different tasks.

MTL methods are used when the tasks are related, meaning that these problems share relevant representations. Whether two tasks are related and can benefit from MTL is a problem without clear answer. Several works have been conducted on task relatedness. Standley et al. [32] proposed a method to identify the most related tasks within a set of tasks, in order to train the related ones in the same architecture, while restricting non-related tasks to be trained by separate networks. Task relatedness is important because some tasks may have conflicting requirements, thus improving one might hurt another, causing overall degradation, a phenomenon known as negative transfer. Negative transfer can occur when the tasks are not related to each other or when the tasks are related to each other, but the MTL approach used is not suitable for the specific problems.

Therefore, tasks that are trained jointly in an MTL framework should share common features in order to improve performance and generalization across domains. A notable example of tasks that have similar spatio-temporal representations is action and hand gesture recognition. Action recognition (AR) focuses on understanding whole-body movements and possibly their interaction with the environment. On the other hand, gesture recognition (GR) aims to interpret movements of specific body parts, such as hands, that convey non-verbal information for communication or interaction. Certain hand gestures might exhibit distinctive hand shapes or specific finger positions that distinguish them from other expressions.

Proc. IEEE/CVF International Conference on Computer Vision (ICCV) Workshops,
Workshop on Affective Behavior Analysis in-the-wild (ABAW), Honolulu, Hawaii, Oct. 2025.

These human-centric tasks analyze human body parts movement to interpret human behavior and intentions thus making them closely related. In this paper, we propose a multi-task learning approach to handle action and gesture recognition problems jointly and show that this method improves the performance and efficiency for both tasks. To the best of our knowledge, there is no deep learning architecture that handles the recognition of both actions and gestures in the same multitask learning framework. In summary, the main contributions of this paper are the following :

- We show that existing deep learning architectures that target action or gesture recognition can be modified using multi-task learning methods to target both tasks, achieving better results in each one of them.
- We evaluate the effects of different multi-task learning methods on the joint training problem of action and gesture recognition.
- We analyze how various multi-task loss calculation methods impact the different multi-task learning methods.

2. Related work

2.1. Multi-Task Learning

In computer vision, most research work on MTL focuses on tasks involving static images. State-of-the-art MTL methods usually formulate MTL as a Single Objective Optimization (SOO) problem to train deep learning architectures to multiple tasks. For instance, Taghavi et al. [33] suggested a shared encoder-decoder architecture, the SwinMTL, to jointly handle depth estimation and semantic segmentation, employing a weighted sum of the loss of each task, so the MTL problem will be addressed as an SOO. On the contrary, MTL can be approached as a Multi-Objective Optimization (MOO) problem, as demonstrated by Sener and Koltun [29], who employed this approach to handle various deep learning tasks, including digit classification, scene understanding, and multi-label classification. A similar approach proposed by Kokkinos [12] is UberNet, a network that handles low-, mid- and high level vision tasks, such as boundary and object detection, semantic segmentation and others, in unified architecture.

Recently, transformer-based approaches have been employed in MTL, due to their ability to capture long-range dependencies across different tasks, owing to their multi-head attention mechanism. Bhattacharjee et al. [1] proposed an end-to-end Multitask Learning Transformer framework called MulT to learn multiple high-level vision tasks simultaneously. Hu and Sign [10] proposed the UniT, a Unified Transformer model that addresses tasks from different domains, including object detection, vision-and-language reasoning and natural language understanding.

2.2. Action Recognition

In action recognition, Convolutional Neural Networks (CNNs) have been the most popular approach to process spatial information, while in order to capture temporal dynamics models that integrate 2D convolutions with sequence processing architectures, such as Long Short-Term Memory (LSTM) networks, have also been extensively used. Moreover, action recognition may benefit from leveraging multimodal data. An example is the work of Rodomagoulakis et al. [28] that integrated audio and RGB video to recognize actions in the context of an HRI system. Visual transformers have recently emerged as a promising alternative to CNN-based architectures, demonstrating competitive performance on benchmark datasets. State-of-the-art models in action benchmarks employ Visual Transformers (ViTs) [4] to extract spatio-temporal information. Lu et al. [21] proposed the Four-Tiered Prompts, which implements a Visual Transformer with a Vision Language Model (VLM) [15] to benefit from their complementary strengths, as ViTs do not generalize well across different domains and VLMs are unable to process videos, achieving state-of-the-art performances on action benchmarks. Similarly, Wang et al. [35] proposed a video masked autoencoder (VideoMAE), which uses a ViT as backbone, achieving state-of-the-art performance on Kinetics and Something-Something [8] benchmarks.

2.3. Gesture Recognition

In gesture recognition, CNN-based architectures have also been the most popular approach. Current state-of-the-art methods in gesture recognition focus on exploiting features extracted across different modalities, such as depth and pose [23]. Köpüklü et al. [13] proposed a CNN-based architecture that fused optical flow and color modalities to achieve competitive performance on Jester and ChaLearn benchmarks. Zhou et al. [38] proposed an architecture to leverage cross-modal spatiotemporal information in RGB-D data, achieving state-of-the-art performance. Transformer-based approaches have also been proposed to be used for gesture recognition, but these works implement a hybrid architecture combining a transformer-based model for the temporal feature analysis and a CNN-based network for spatial feature extraction [7].

2.4. Multi-task Learning in Action and Gesture Recognition

Multi-task learning has also been applied to action and gesture recognition separately. In action recognition, Luvizon et al. [22] proposed an MTL framework that could handle 2D or 3D pose estimation from images and classify human actions from video sequences. Furthermore, action recognition has been combined with other tasks, such as saliency estimation and video summarization [14]. Also,

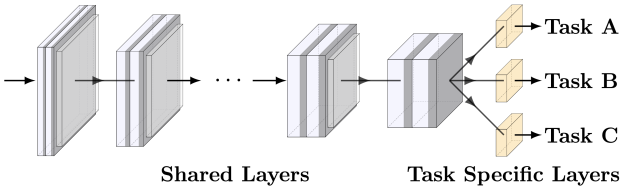

Figure 1. An instance of a hard parameter sharing model for three tasks. The first layers of the model (gray color) are common for all the tasks, while the last layers (yellow color) are task-specific.

Simonyan and Zisserman [30] utilized a framework, where the UCF-101 and HMDB-51 benchmarks, both containing action videos, were handled as separate tasks in a multi-task learning network. In gesture recognition, Fan et al. [6] proposed a MTL framework to handle gesture recognition and segmentation. A CNN was used to learn segmentation information with depth modality supervision during the training process, while requiring only the RGB modality during inference. Therefore, to the best of our knowledge, this paper is the first to propose an MTL framework that handles actions and gestures jointly.

3. Methodology

In this work, we propose a way to adjust a deep learning architecture to jointly handle action and gesture recognition using MTL methods, while showing that this approach benefits both tasks. We evaluate the proposed method with several experiments exploring the effect of two key factors: the weight sharing method and the multi-task loss calculation. The weight sharing method refers to the way the MTL model defines which parameters will be allocated by the different tasks, enabling or preventing information sharing between them, while the multi-task loss calculation refers to the way the overall loss across all tasks is calculated. To do so, a deep learning architecture used as a backbone is altered according to the different weight sharing methods. In this work we use ResNet-3D [9] as the backbone, due to its performance in many action and gesture benchmark datasets. However, other popular 3D-CNNs could be used.

The most commonly used approach in MTL is the Hard Parameter Sharing (HPS) [3], due its simplicity and effectiveness. Figure 1 demonstrates how a deep learning architecture can be converted into an HPS model. All the network's structure is retained except for the last layers, which are duplicated to create task-specific layers to match the output of the specific tasks. By using the same layers, the model jointly tunes these parameters across tasks to share information between them.

Another popular weight sharing method is soft parameter sharing (SPS) [37], which allows the architecture to control the amount of information shared between tasks. In SPS, each task is assigned a backbone model and information sharing is achieved through connection units between these task-specific networks, as illustrated in Figure 2. The most widely used SPS architectures are Cross-Stitch Networks [25], whose connection units, referred to as cross-stitch units, control the information passed through to the next layer of the model by linearly combining the output of the shared layers. Although, they can be placed anywhere in the network, better performance is empirically observed after the pooling activation maps.

At each layer of the network, these units learn a linear combination of the activation maps of the tasks. For two activation maps x_A, x_B from layer l the cross-stich unit learns linear combinations $\tilde{x}_A$, $\tilde{x}_B$, parameterized by h. For location (i, j) in the activation map, these are given by:

$$\tilde{\mathbf{x}}^{ij} = \mathbf{H}\mathbf{x}^{ij} \Rightarrow \begin{bmatrix} \tilde{x}_A^{ij} \\ \tilde{x}_B^{ij} \end{bmatrix} = \begin{bmatrix} h_{AA} & h_{AB} \\ h_{BA} & h_{BB} \end{bmatrix} \begin{bmatrix} x_A^{ij} \\ x_B^{ij} \end{bmatrix}$$

where h_{AA} and h_{BB} are the same-task values, since they weigh the activations of the same task, while h_{AB} and h_{BA} are the different-task values, since they weigh the activations of another task. In practice a hyperparameter s, can be used to represent the percentage of the information shared between the task-specific networks. By varying the value of s, the unit can decide between shared and task-specific representations, or choose a middle ground.

The term MTL Layer is used to describe all the layers of the different task specific networks at a certain level of the model, as can be seen by the blue dashed rectangle in Figure 2. HPS and SPS methods both share information across all MTL layers of the network. In HPS information is shared completely across an MTL layer, while in SPS the amount of shared information in an MTL layer is regulated by a factor, which is the same for all the MTL layers. In other words, all the MTL layers in both HPS and SPS share the same amount of information without considering that different layers in a neural network learn distinct types of features. Early layers typically capture more general features, while deeper layers learn more complex representations relevant to the task addressed. Consequently, an MTL architecture can benefit from regulating each layer's contribution to a specific task's inference. The Learned Weight Sharing (LWS) method, introduced by Prellberg et al. [27], addresses this issue by searching for the optimal amount of shared information at a specific MTL layer for each task.

Specifically, the LWS architecture is constructed by duplicating a backbone model once for each task. If the backbone model has a total of N layers and the LWS model handles K tasks, then the multi-task model has $K \times N$ different layers, each layer having their own set of parameters. For instance, in Figure 3 an LWS architecture handling two tasks is depicted, with each task represented with different colors. Initially, each layer is assigned to the task that the corresponding backbone model originally addressed. Dur-

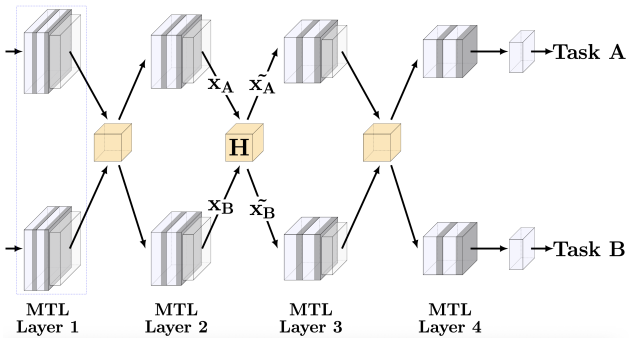

Figure 2. Cross Stitch Units applied on two task specific CNNs. The task specific models, illustrated with gray color, are connected with units, depicted with yellow color, which control the information shared between the two tasks. The term MTL Layer is used to describe all the layers of the different task specific networks at a certain layer of the model.

ing training, each MTL layer of the architecture, except for the last task specific layer, is compatible with both tasks and can be trained on samples of either task. In this method information sharing is achieved by training the same layer on all different tasks according to their respective learned probabilities.

In addition, the LWS algorithm adjusts the initial assignments, resulting in a set of optimal layer-task combinations that is used during inference, while simultaneously training the weights themselves. To achieve this, training is accomplished by two optimization algorithms: the Natural Evolution Strategy (NES) and the Stochastic Gradient Descent (SGD). In particular, the NES optimizer [36] is used for the layer-task assignment problem and the SGD optimizer is used for weight optimization.

The search for optimal assignments and layer weights is formulated as an optimization problem, expressed as follows:

$$\min_{\theta,\alpha} f(\theta, \alpha),$$

where $f : \Theta \times A \rightarrow R$ is the loss over all tasks, $\theta \in \Theta$ is a vector of all layer weights, and $\alpha \in A$ is an assignment of weights to task-specific layers. The loss function f is differentiable wrt. θ, but non-differentiable wrt. α, as the assignment of a certain layer to a task-specific network is a discrete problem. LWS solves a stochastic version of the problem by introducing a probability distribution π over the set of all possible assignments of weights to task-specific layers A with a probability density function $p(\alpha|\pi)$. Thus, the optimization problem is described as follows:

$$\min_{\theta,\pi} J(\theta, \pi) = E_{\alpha \sim p(\alpha|\pi)}[f(\theta, \alpha)].$$

This stochastic formulation transforms the discrete, non-differentiable optimization problem over the assignments α into a continuous, differentiable optimization problem over the parameter π. The optimization problem is then solved by alternating between an assignment optimization step and a weight optimization step.

For the assignment optimization, θ is fixed and the assignments of weights $a_1, ..., a_{\lambda_\pi}$, distributed according to $p(\alpha|\pi)$, are sampled. Their loss values $l_i = f(\theta, \alpha_i)$ are calculated on the same batch of training data for all assignments. A Monte-Carlo approximation, with population size λ, of the gradient of the loss function wrt. π is computed as follows:

$$\nabla_\pi J(\theta, \pi) \approx \frac{1}{\lambda_\pi} \sum_{i=1}^{\lambda_\pi} u_i \nabla_\pi \log p(\alpha_i|\pi),$$

where u_i denotes the utility values, which are created by fitness shaping, a method commonly used to transform raw scores, such as loss values, into a regularized range of utility values. This approach makes the algorithm invariant to the scale of the loss function, as it focuses on the relative ranking of the loss values. Specifically, in the LWS method, the utility values are calculated with the following formula:

$$u_i = 2 \cdot \frac{l_i - 1}{\lambda_\pi - 1} - 1.$$

The gradient is then used to update the parameters of the probability distribution π with learning rate η_π, according to the following step in the direction of $\nabla_\pi J(\theta, \pi)$:

$$\pi + \eta_\pi \nabla_\pi J(\theta, \pi).$$

For the weight optimization, while keeping π fixed, the assignments of weights to task-specific network layers $a_1, ..., a_{\lambda_\theta}$ distributed according to the pdf $p(\alpha|\pi)$ are sampled and backpropagation is performed for each sample. The same batch of training data is used for the backpropagation step throughout this process for every assignment. The resulting gradients $\nabla_\theta f(\theta, \alpha_i)$ are averaged over all assignments, so that the final gradient is described by the following equation:

$$\nabla_\theta J(\theta, \pi) \approx \frac{1}{\lambda_\theta} \sum_{i=1}^{\lambda_\theta} \nabla_\theta f(\theta, \alpha_i).$$

Using this gradient, θ is updated by SGD with learning rate η_θ, according to the following step:

$$\theta - \eta_\theta \nabla_\pi J(\theta, \pi).$$

In the LWS algorithm, during training, the assignment of weights per layer is sampled over the probability density function. At the beginning of the training phase the probability of assigning a certain layer to a specific task is equally distributed and the parameters of each layer are tuned using data from all the tasks, learning more general features. As

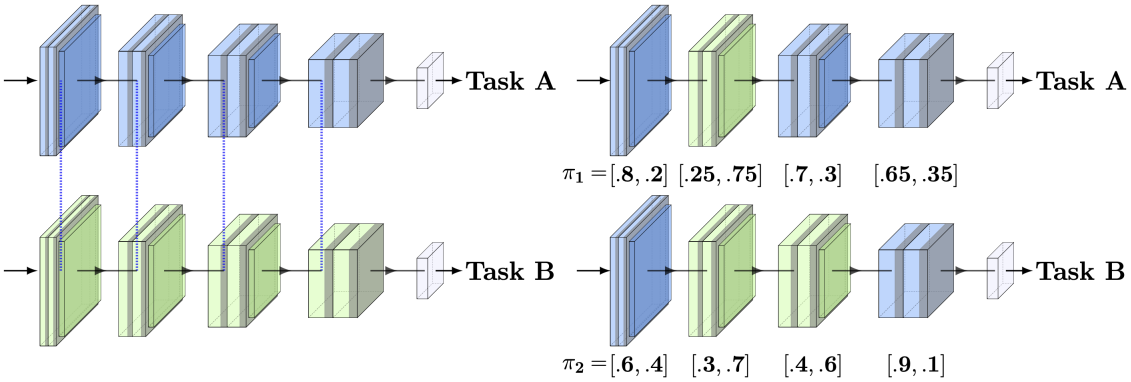

Figure 3. *Left*: Representation of an LWS architecture for two tasks. Layers between task-specific networks are compatible for all tasks, so the model learns which parameters will be used in each layer for each task. *Right*: During training, the model searches for the optimal assignment of weights of layers per task-specific network and updates the probability that certain layer weights are used by a specific task. When the same set of weights in a layer is used for training on both tasks, information sharing is achieved. During inference, the most probable assignment of weights in each layer is used for each task-specific network.

training progresses, the probability distribution is optimized to better fit the MTL problem, thus the layers used to process a sample of a certain task are more related to this task. At this stage, layers that have been tuned on unrelated tasks are less likely to be assigned to the task-specific network for a given task.

On the other hand, during inference the model selects the most probable set of weights for each layer and constructs the task-specific network, as depicted in Figure 3. This network achieves optimal results, as the weights chosen are specific to the task addressed. Since different MTL layers require varying amounts of weight sharing, the layers of the model control the information shared across tasks. The first layers encode more robust and generalizable features, benefiting from training on multiple tasks. In contrast, the final layers learn more specific features and therefore information sharing between tasks is limited.

Another important factor to consider when designing a MTL architecture is how the multi-task loss is calculated. The multi-task loss refers to the total loss composed of the individual losses from all tasks. The simplest way to calculate the multi-task loss is by averaging the losses of all tasks, which treats all tasks equally. However, this does not take into account the difficulty of each task, which can lead to suboptimal results. The mathematical formulation of the average loss calculation is:

$$\mathcal{L}_{MTL} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i,$$

where N is the number of tasks and $\mathcal{L}_i$ is the loss of the i-th task.

To consider the difficulty of each task, weights are assigned to task specific losses. The performance of the model depends heavily on the selection of weights, while searching for these optimal values is computationally expensive, especially for large models with numerous tasks. Kendall et al. [11] proposed an uncertainty-based method for computing the multi-task loss that learns task weights during training, since task importance is not known in advance. Specifically, task loss weights are trainable parameters integrated into the objective function that describes the model. The mathematical formulation of the uncertainty loss calculation is:

$$\mathcal{L}_{MTL}(\boldsymbol{\sigma}) = \sum_{i=1}^{N}\frac{1}{2\boldsymbol{\sigma}^2}\mathcal{L}_i + \log(\boldsymbol{\sigma})),$$

where $\sigma_i > 0$ are the uncertainty weights for the two tasks. The magnitude of these parameters determines how uniform the discrete distribution is.

Although the uncertainty in weighing tasks provided a way to determine the relative importance of the tasks, in practice it resulted in negative values, which is not acceptable in the context of uncertainty estimation. To address this issue Liebel et al. [16] altered the uncertainty loss calculation to avoid negative values for the variance, by converting the regularization term so that only values greater than one were allowed in the logarithm. This is referred to as automatic weight loss and it is calculated as follows:

$$\mathcal{L}_{MTL}(\boldsymbol{\sigma}) = \sum_{i=1}^{N}\frac{1}{2\boldsymbol{\sigma}^2}\mathcal{L}_i + \log(1+\boldsymbol{\sigma}^2)).$$

The uncertainty and its extension to automatic loss calculation method leverages probabilistic modeling to learn weights based on how noisy a task is. This loss calculation method is more suitable for videos in-the-wild which are

typically noisy, with large variations in environmental settings, as opposed to videos acquired in a lab environment.

Some other MTL methods prefer to focus on how each task performs during training, rather than the environmental setting in their data. A notable example is the Dynamic Weight Average (DWA) loss calculation [20], which assigns a weight to each task based on the rate of change of loss for each task during previous iterations. For the implementation, only the numerical values of the losses of each task at the current and previous iterations are required. In practice, DWA assigns higher weights to tasks with lower loss rates, which means that the model will focus more on tasks that improve more slowly, in order to achieve better overall performance. The DWA loss is calculated as follows:

$$\mathcal{L}_{MTL} = \sum_{i=1}^{N} \lambda_i(t)\mathcal{L}_i,$$
$$\text{where } \lambda_i(t) = \frac{K \exp(\mathbf{w_i}(t-1)/T)}{\sum_k \exp(\mathbf{w_k}(t-1)/T)},$$
$$\mathbf{w_i}(t-1) = \frac{\mathcal{L}_i(t-1)}{\mathcal{L}_i(t-2)} \ .$$

L_i is the loss of the i-th task, $\lambda_i(t)$ is the weight of the i-th task at time t, $w_i(t-1)$ is the weight of the i-th task at time $t-1$, T is a scaling factor that controls the softness of task weighting, with large values resulting in a more even distribution between different tasks and K is the number of tasks.

In the implementation of the DWA loss, the loss $L_i(t)$ is calculated as the average loss over several iterations, so it reduces the uncertainty from stochastic gradient descent and random training data selection. For $t = 1, 2$ the weights $w_k(t)$ are initialized to 1, but any non-balanced initialization based on prior knowledge could also be introduced.

The choice of the optimal multi-task loss calculation method for a specific MTL problem is not trivial and in most works it is experimentally determined. To find the optimal loss calculation method, we experiment with multiple sets of data from diverse sources, with varying amounts of noise and different environmental settings.

4. Experiments

4.1. Datasets

In this paper, we use datasets with action samples from the UCF-101 and NTU-RGB+D datasets and gesture samples from the IsoGD and NVGesture benchmarks. Specifically, UCF-101 [31] consists of 13,320 videos divided in 101 categories with realistic actions, since the data is collected from YouTube. While 3 different train-test splits are proposed, in our experiments we use the split-1. NTU-RGB+D [18] is a large-scale dataset for human action recognition, containing 114,480 samples of 120 action classes performed by 106 subjects. In our experiments, we use the cross-subject proposed split to evaluate the models. IsoGD [34] is a large-scale dataset for RGB-D gesture recognition. The dataset contains 47,933 RGB-D gesture videos, with 249 gesture labels performed by 21 different individuals. NVGesture [26] is a smaller hand gesture dataset captured composed of 1,532 dynamic gestures performed by 20 subjects, categorized into 25 classes. Moreover, we used pretrained weights for our models trained on the Kinetics-400 [2] dataset. Kinetics-400 is a large-scale human action dataset with videos collected from YouTube. It consists of around 240,000 video clips covering 400 human action classes with at least 400 video clips for each action class.

In the MTL methods, chosen in this paper, the input samples alternate between the tasks addressed. Therefore, we implemented a Multi-task Dataset according to the MTL dataset structure in [27], so that each training batch contains samples of all tasks. For the subsets of actions, samples from the datasets UCF-101 and NTU-RGB+D 120 are chosen and for the subsets of gestures, samples from the datasets IsoGD and NVGesture are used.

The NTU-RGB+D dataset contains classes, such as hand waving or clapping, which can be considered as gestures, since they consist mostly of hand motions. So, in order to properly evaluate the proposed MTL approach between actions and gestures, the NTU-RGB+D dataset is split into two parts: one containing only the action classes (NTU_AR) and the other containing only the gesture classes (NTU_GR). The separation in action and gesture sets is done manually. The NTU_AR set is considered as the action task, while NTU_GR set is merged with the gesture samples from the IsoGD and NVGesture to be handled as the gesture task, resulting in the sets NTU_GR_IsoGD and NVGesture NTU_GR_NVGesture. When combining the gesture classes from the NTU-RGB+D with IsoGD and NVGesture, we merge overlapping gesture classes into one.

So, the experiments were conducted on four different sets of action-gesture datasets. Each set has different proportion of action and gesture classes, as well as number of samples used for each task. The distribution of classes and samples of the multi-task action and gesture sets is depicted in Figure 4. Set-1, which is constructed from the UCF-101 and the IsoGD datasets, consists mostly of gestures, as approximately 75% of all classes and samples belong to them. The opposite is true for the Set-2, which has samples from UCF-101 and NVGesture. For this set the majority of the classes and the samples belong to actions (80% and 88% respectively). In Set-3, which consists of action samples from NTU_AR set and gesture samples from NTU_GR_IsoGD, although most of the classes are gestures (74%), the action samples (65%) are about twice the samples of gestures (35%). Set-4, which is made from samples from NTU_AR and NTU_GR_NVGesture, consists primar-

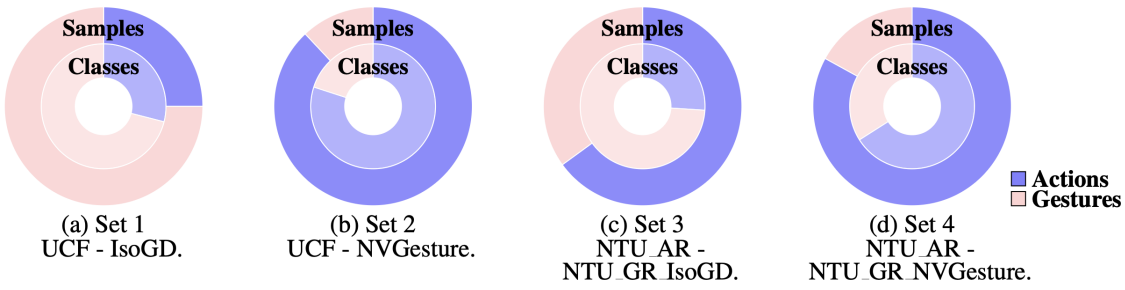

Figure 4. Classes and samples distribution across different multi-task sets of action and gesture classes. The inner circle represents the classes of each set and the outer circle represents the total number of samples used. (a) Set-1 is constructed from the UCF-101 and the IsoGD datasets. (b) Set-2 has samples from UCF-101 and NVGesture. (c) Set-3 consists of action samples from NTU_AR set and gesture samples from NTU_GR_IsoGD. (d) Set-4 is made from samples from NTU_AR and NTU_GR_NVGesture.

ily of action classes and samples, with about 66% and 83% respectively.

4.2. Implementation Details

The model used as the backbone for the MTL networks is a ResNet-3D with 18 layers pretrained on the Kinetics-400 dataset. We chose this model due to its efficiency in various computer vision tasks. We left its last 2 Residual Layers trainable. This structure helped us evaluate the proposed MTL method, while also benefit from the quicker convergence [24].

MTL networks have more trainable parameters than STL variants, since they have some additional task-specific layers to handle the different tasks. The method that has comparable trainable parameters with the STL variants is the HPS, because only the last layer uses task-specific layers. On the other hand, the other two methods (SPS and LWS) have approximately twice the number of total trainable parameters of the STL model, due to the duplicated backbone model in their architecture. So, we also implement a STL ResNet-3D with 34 layers pretrained on the Kinetics-400 dataset, which has approximately the same trainable parameters as the SPS and LWS models.

Both MTL and STL networks are trained for a specific number of iterations, instead of epochs, due to the different number of samples each set has. In each iteration, a batch of samples, that contains both actions and gestures, is drawn from the multi-task dataset and passes through the model. This training scheme allows the model to process the same number of samples in each training, so the performance of the MTL methods can be compared with the STL methods. The number of iterations per training is empirically fixed to 20,000, because all the evaluated models have reached convergence at that point.

The choice of the optimizer plays a significant role for multi-task learning problems. Many works on MTL setups have used the Adam optimizer [17] while others have trained MTL architectures using the SGD optimizer [19]. Elich et al. [5] have compared these two optimizers and empirically reached the conclusion that the Adam optimizer performs better than the SGD optimizer for MTL networks. So, in our experiments we used Adam optimizer for all the STL and MTL experiments. Also, we used batch size of 8 and the videos are downsampled to a fixed length of 32 frames per video and center cropped to have dimensions of 112x112 pixels.

4.3. Results

In Table 1, the per task results and the cumulative accuracy for different parameter sharing methods and multi-task loss calculation methods on the various MTL datasets are presented. The cumulative accuracy is calculated as the percentage of all correct predictions to the number of total predictions across all tasks. We observe that almost all of the MTL experiments outperform their STL variants, providing prominent evidence that MTL is effective at leveraging common patterns present in actions and gestures, enhancing the recognition of both. We evaluate the results regarding three factors: the weight sharing method, the multi-task loss calculation method and the distribution of actions and gestures in the multi-task datasets. Also, we compare the MTL networks to 3D-ResNet-34, which has similar trainable parameters as the SPS and LWS multi-task models.

All weight sharing methods outperform their STL variants, except for the case of the HPS models trained on Set-1, indicating that multi-task learning can be beneficial to simultaneously train a model on actions and gestures. A possible reason for the STL models to perform better than HPS in Set-1 might be the fact that HPS models share their parameters completely, failing to learn task-specific representations. Set-1 consists mainly of gestures and when a hand gesture is performed, the spatial range of a gesture is limited to a specific body part, making the model less capable to generalize to tasks that involve whole body movements.

Learning	Loss	Set-1 (%)			Set-2 (%)			Set-3 (%)			Set-4 (%)		
		Actions	Gestures	All	Actions	Gestures	All	Actions	Gestures	All	Actions	Gestures	All
STL (R18)	-	58.26	25.67	37.93	58.26	20.54	53.99	61.39	41.35	55.41	61.39	68.84	63.02
STL (R34)	-	60.59	28.31	40.45	60.59	19.78	55.98	64.53	48.06	59.62	64.53	65.69	64.78
HPS	AVG	54.16	20.49	33.16	64.34	27.86	60.21	70.65	60.25	67.55	71.91	74.33	72.44
HPS	DWA	57.36	25.45	37.45	58.6	25.57	54.86	64.45	58.87	62.78	72.53	73.85	71.81
HPS	AUT	60.53	20.02	35.25	66.24	33.68	62.55	49.42	56.57	51.55	72.51	72.80	72.57
SPS (s=.2)	AVG	67.64	26.6	42.04	69.94	36.17	66.12	72.27	60.13	68.65	74.2	74.49	74.26
SPS (s=.2)	DWA	62.2	30.96	42.71	71.66	37.42	67.78	71.84	60.91	68.57	73.51	74.59	73.74
SPS (s=.2)	AUT	67.62	26.25	41.81	71.66	43.24	68.44	74.45	62.14	70.77	74.17	73.9	74.11
SPS (s=.8)	AVG	64.22	26.78	40.85	**73.14**	**35.76**	**68.91**	**74.2**	**74.49**	**74.28**	72.65	74.21	72.99
SPS (s=.8)	DWA	63.1	32.03	43.72	67.94	34.93	64.19	73.51	74.59	73.83	**74.03**	**75.51**	**74.35**
SPS (s=.8)	AUT	**70.24**	**35.79**	**48.74**	68.75	39.09	65.39	73.66	60.8	69.82	72.25	73.54	72.53
LWS	AVG	64.39	35.2	46.18	69.73	39.62	66.33	73.03	56.27	68.03	73.45	74.43	73.66
LWS	DWA	70.58	34.9	48.32	71.42	43.04	68.21	72.6	53.64	66.94	73.41	75.47	73.86
LWS	AUT	64.79	28.7	42.28	72.27	37.21	68.30	72.48	53.07	66.68	72.96	76.4	73.71

Table 1. Multi-task Learning results for three different parameter sharing methods (HPS - hard parameter sharing, SPS - soft parameter sharing, LWS - learned weight sharing) and three different loss calculation methods (AVG - average, DWA - dynamic weight average, AUT - automatic) on different multitask datasets (Sets 1-4) containing action and gesture videos. Sets 1-4 are constructed by combining single-task action and gesture datasets (see Figure 4). We compare MTL and STL by their accuracy on Actions, Gestures and on all samples together (ALL). For MTL methods, we used the 3D-ResNet-18 model as the backbone, while for STL, we implemented both 3D-ResNet-18 (R18) and 3D-ResNet-34 (R34).

When using a set consisting mostly of gestures, MTL methods such as SPS and LWS should be used to control the amount of information shared, to help the model learn more task-specific features.

For the choice of multi-task loss calculation we observe that the automatic method seems to achieve better results when paired with SPS methods, while average and dwa provide promising results with either SPS or LWS approaches. In HPS method, there is no clear selection for the multi-task loss calculation method, as it seems to depend on the samples of the data used.

Regarding the different ratios of action and gesture samples in the multi-task dataset, it is evident that MTL methods favor sets that contain more actions than gestures. On the other hand, if a set contains more gestures than actions, more complex sharing methods, such as SPS and LWS, can still benefit from multi-task learning approaches.

When it comes to model size, STL-3D-ResNet-34, SPS-3D-ResNet-18 and LWS-3D-ResNet-18 have approximately twice the number of trainable parameters compared to STL-3D-ResNet-18 and the HPS-3D-ResNet-18. The results indicate that MTL models with more shallow backbone architectures can still outperform deeper single-task networks. All HPS models have half the trainable parameters from STL-3D-ResNet-34 and still achieve better results in most cases. On the other hand, while SPS and LWS models have similar number of trainable parameters compared to STL-3D-ResNet-34 still perform better than this deeper architecture.

Overall, SPS achieves the best results across all weight sharing methods for the joint training of action and gesture recognition, while the optimal choice of the multi-task loss calculation method depends on the distribution of actions and gestures in the dataset.

5. Conslusion

Action and gesture recognition are key aspects of human behavior analysis. In this work, we show that multi-task learning is an effective way to address both problems using a single deep learning architecture. We experimented with different weight sharing approaches, multi-task loss calculation methods and with diverse sets of multi-task datasets. We demonstrated that almost all multi-task approaches outperform their single task variants. Thus we provide strong evidence that a multi-task learning paradigm for action and gesture recognition captures more generalizable visual representations leading to more efficient and robust models, which are essential for practical applications.

Acknowledgement

This project was partially funded by the European Union under Horizon Europe (grant No. 101136568 - HERON).

Funded by the European Union

References

[1] Deblina Bhattacharjee, Tong Zhang, Sabine Süsstrunk, and Mathieu Salzmann. Mult: An end-to-end multitask learning transformer. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[2] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 6

[3] Rich Caruana. Multitask learning. *Machine Learning*, 28: 41–75, 1997. 1, 3

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, G Heigold, S Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2020. 2

[5] Cathrin Elich, Lukas Kirchdorfer, Jan M Köhler, and Lukas Schott. Examining common paradigms in multi-task learning. In *Proc. of the DAGM German Conference on Pattern Recognition (GCPR)*, 2024. 7

[6] Dinghao Fan, Hengjie Lu, Shugong Xu, and Shan Cao. Multi-task and multi-modal learning for rgb dynamic gesture recognition. *IEEE Sensors Journal*, 21(23):27026–27036, 2021. 3

[7] Mallika Garg, Debashis Ghosh, and Pyari Mohan Pradhan. Gestformer: Multiscale wavelet pooling transformer network for dynamic hand gesture recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[8] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense. In *Proc. of the IEEE International Conference on Computer Vision (ICCV)*, 2017. 2

[9] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Learning spatio-temporal features with 3d residual networks for action recognition. In *Proc. of the IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017. 3

[10] Ronghang Hu and Amanpreet Singh. Unit: Multimodal multitask learning with a unified transformer. In *Proc. of the IEEE International Conference on Computer Vision (ICCV)*, 2021. 2

[11] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 5

[12] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[13] Okan Kopuklu, Neslihan Kose, and Gerhard Rigoll. Motion fused frames: Data level fusion strategy for hand gesture recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018. 2

[14] Petros Koutras and Petros Maragos. Susinet: See, understand and summarize it. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019. 2

[15] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proc. of the International Conference on Machine Learning (ICML)*, 2023. 2

[16] Lukas Liebel and Marco Körner. Auxiliary tasks in multitask learning. *arXiv preprint arXiv:1805.06334*, 2018. 5

[17] Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 34:18878–18890, 2021. 7

[18] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C Kot. Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2684–2701, 2019. 6

[19] Liyang Liu, Yi Li, Zhanghui Kuang, J Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2021. 7

[20] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 6

[21] Hui Lu, Hu Jian, Ronald Poppe, and Albert Ali Salah. Enhancing video transformers for action understanding with vlm-aided training. *arXiv preprint arXiv:2403.16128*, 2024. 2

[22] Diogo C Luvizon, David Picard, and Hedi Tabia. Multi-task deep learning for real-time 3d human pose estimation and action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(8):2752–2764, 2020. 2

[23] Yujun Ma, Benjia Zhou, Ruili Wang, and Pichao Wang. Multi-stage factorized spatio-temporal representation for rgb-d action and gesture recognition. In *Proc. of the 31st ACM International Conference on Multimedia (ACM MM)*, 2023. 2

[24] Tahir Mehmood, Alfonso E Gerevini, Alberto Lavelli, and Ivan Serina. Combining multi-task learning with transfer learning for biomedical named entity recognition. *Procedia Computer Science*, 176:848–857, 2020. 7

[25] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[26] Pavlo Molchanov, Xiaodong Yang, Shalini Gupta, Kihwan Kim, Stephen Tyree, and Jan Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6

[27] Jonas Prellberg and Oliver Kramer. Learned weight sharing for deep multi-task learning by natural evolution strategy and stochastic gradient descent. In *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, 2020. 3, 6

[28] Isidoros Rodomagoulakis, Nikolaos Kardaris, Vassilis Pitsikalis, Effrosyni Mavroudi, Athanasios Katsamanis, Antigoni Tsiami, and Petros Maragos. Multimodal human action recognition in assistive human-robot interaction. In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016. 2

[29] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in Neural Information Processing Systems*, 31, 2018. 2

[30] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in Neural Information Processing Systems*, 27: 568–576, 2014. 3

[31] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild. Technical Report CRCV-TR-12-01, Center for Research in Computer Vision (CRCV). November, 2012. 6

[32] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *Proc. of the International Conference on Machine Learning (ICML)*, 2020. 1

[33] Pardis Taghavi, Reza Langari, and Gaurav Pandey. Swin-mtl: A shared architecture for simultaneous depth estimation and semantic segmentation from monocular camera images. In *Proc. of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2024. 2

[34] Jun Wan, Yibing Zhao, Shuai Zhou, Isabelle Guyon, Sergio Escalera, and Stan Z Li. Chalearn looking at people rgb-d isolated and continuous datasets for gesture recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2016. 6

[35] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. Videomae v2: Scaling video masked autoencoders with dual masking. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2

[36] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15 (1):949–980, 2014. 4

[37] Yongxin Yang and Timothy Hospedales. Trace norm regularised deep multi-task learning. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2017. 3

[38] Benjia Zhou, Pichao Wang, Jun Wan, Yanyan Liang, Fan Wang, Du Zhang, Zhen Lei, Hao Li, and Rong Jin. Decoupling and recoupling spatiotemporal representation for rgb-d-based motion recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2