Sign Language Alphabet Translator Using Transfer Learning with Object Detection

Yi-Chuan Leuker, Agostino Calamia, Torsten Wolter

yi-chuan.leuker@fom-net.de, agostino.calamia@fom-net.de, torsten.wolter@fom-net.de

Abstract

Every country has its own sign language. There is no universal sign language in the world to help deaf people communicate with others from other countries. This project focuses on developing a Sign Language Alphabet (SLA) translator that can play an important role by not only its interpretation but also helping deaf people to communicate with each other without learning a new sign language. In this project, the sign language used for training detective models is American sign language. The detective models were imported from pre-trained models for transfer learning in the Keras library. *VGG16*, *ResNet50V2*, *MobileNet*, *MobileNetV2*, *DenseNet201*, and *Xception* were selected as top performance models from each architecture by comparing the high accuracy performance with the partial learning for further processing training of the entire training dataset. After training the full training dataset, these six models were compared for accuracy, training duration, and inference time. The result showed that *MobileNet* and *MobileNetV2* have similar real-time capabilities: approx. 99% of accuracy and approx. 0.03 seconds of inference time. In the end, the *MobileNet* was successfully implemented with a test dataset to confirm the usability of translating to Turkish sign language images.

Index terms should be included, as shown below.

Index Terms: sign language alphabet recognition, transfer learnings, object detection

1. Introduction

The World Health Organization (WHO) projected that nearly 2.5 billion people in the world have hearing loss by 2050 and emphasized that sign language-related applications are essential tools for deaf people [1].

Instead of oral language using speaking and listening, Sign language is a form of visual communication through gestures, body movements and facial expressions. Sign language can be used for different purposes in different situations, but it is primarily designed for communication with the deaf. As it is more visually accessible to the deaf, sign language is a natural method of communication for the deaf people and can express a wide range of meanings in the same way as spoken language; for example, American Sign Language is used by deaf people in the United States and partial provinces of Canada, British Sign Language is used by deaf people in the UK, French Sign Language in France, Japanese Sign Language in Japan and Chinese Sign Language in China.

Although there is no unified international sign language yet, the major sign language systems such as the USA, British, France and China have developed fingerspelling, also called "Sign Language Alphabet" (SLA). SLA represents the 26 common alphabets, A to Z, and local special letters using only hand gestures, mostly in difficult words such as personal names, place names, technical terms, etc. Additionally, it can also be used to facilitate easier communication when encountering issues with sign language dialects.

In the past decade, deep learning has shown excellent performance in image recognition, enabling breakthroughs in sign language recognition technology. Many studies have used neural networks to convert static SLA signs into text or speech, using images of hand gestures as input, and the main approach includes data acquisition technique, static signs, classification with over 90% accuracy of recognition [2]. Moreover, Transfer Learning is one of the major milestones in Deep Learning for Object Detection. With transfer learning, pre-trained models have already been trained on a different task; therefore, it is a short-cut that re-uses the pre-trained model such as its trained weights for a shorter training process. This project aims to utilize pre-trained models from transfer learning with Object Detection in Keras and to apply static hand gesture images into different pre-trained model architecture to present a development of an SLA translator, which translates American SLA to Turkish SLA, thereby helping deaf people to communicate with each other without learning a new sign language. The remainder of this paper will include related work in the sign language recognition research area, the problem statement found in the research, the objective of the project, and the methodology applied in the project. After evaluating the top accuracy rate from each architecture from all the available pre-trained models in the Kera transfer learning with Object Detection library, six pre-trained models, *VGG16*, *ResNet50V2*, *MobileNet*, *MobileNetV2*, *DenseNet201*, and *Xception*, are selected into the short list and trained as predictive models with a full training dataset. The results of relevant accuracy, training duration, and inference time were evaluated as final metrics to determine the most suitable model for the use case in real-time translation. In the end, the most suitable model is used to predict the Turkish SLA images from the ASL images using the test dataset.

2. Related Work

In the last decade, the development of gesture recognition technology has led to a breakthrough in SLA translators. There are two main approaches to sign language recognition: sensor-based or vision-based[3]. The major difference between sensor-based and vision-based approaches is in the data acquisition phase. Sensor-based approaches utilize sensor instruments such as sensory gloves to capture sign language, but such equipment was too complex and expensive to be widely used. On the other hand, vision-based approaches don't require complex facilities to acquire data, acquiring images or videos of the sign language through a camera. For example, D., Cao et al.[4] in 2015 developed sign language recognition by adapting Microsoft Kinect technology and used Random forest to successfully recognized static 24 American SLA with above 90% accuracy. Furthermore, A., Joshi et al.[5] presented a real-time automated American SLA translator that translates American SLA to English text by applying edge detection and cross-correlation methodologies, resulting in 94.23% accuracy for alphabets. The Convolutional neural network (CNN) has become a common method applied in image recognition and classification in recent years. M., Taskiran et al.[6] designed a real-time sign language system with the implementation of feature extraction and classifier based on a CNN structure, resulting in 98.05% accuracy.

3. Research question

3.1. Problem Statement

The previous works mentioned in the previous chapter showed great achievements in translating American SLA to texts by capturing images through cameras and using CNN for feature extraction and classification. However, an SLA translator from American SLA to other SLA is still not available today, which would solve the international communication of people without learning a new SLA. However, the challenge lies in identifying a detective model that delivers both high accuracy and fast inference time.

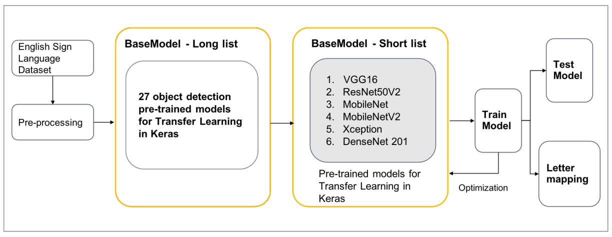
3.2. Objective

This research question aims to develop an SLA translator that uses optimal transfer learning for Object Detection to recognize images of American SLAs and translate them into Turkish SLAs with images. For example, the American letter P in sign language is recognized and translated into the Turkish letter P in sign language. The main focus will be to determine the most suitable model that potentially has both high accuracy and efficient inference time in real-time application.

4. Methodology

Figure 1 shows the proposed architecture to the ASL translator. The main approach to developing a translation from American SLA to other SLA is to build the best image recognition model that recognizes each American letter from the sign language with high accuracy. Instead of learning from scratch, transfer learning with pre-trained models, which were already trained on a large benchmark image dataset, can save a lot of computational cost and help the performance. Keras provides a wide range of pre-trained models for deep learning such as VGG, ResNet, Xception, etc [7]. All the available pre-trained models will be applied into the first run training with the partial dataset to find out the 5 best performing models from each architecture with the highest accuracy of prediction. The top 5 models will be further optimized and trained with all the training data to determine the most suitable ASL recognition model. In the last phase, the results of the prediction will be mapped with Turkish ASL images.

Figure 1: *The proposed approach for the sign language translator*



4.1. Dataset

The American ASL dataset is collected from one of the top open-source dataset repositories, Kaggle. The dataset contains 87,000 images with 26 American letters and 3 extra signs, which are delete, space, and nothing. All the images are equally distributed to the 29 signs. In other words, each sign has 4,300 images. The whole dataset is split into training and test dataset. The training dataset is 80% of the data, and the rest data belongs to the test dataset.

4.2. Image preprocessing

The original image size in the dataset is 200 x 200 pixels, but the network of pre-trained models is trained on 224 x 224 color images. Therefore, the images in the dataset are then resized to 224 x 224 pixels and are at the same time randomly transformed with data augmentation by applying the Keras ImageDataGenerator class [7]. ImageDataGenerator class provides an ability to use data augmentation such as rotations, shift, brightness as well as zoom and return augmented images automatically when training a model to help improve model performance.

4.3. Transfer learning models

One of the main benefits of using transfer learning is to make use of previously trained models and save computational cost (CC) for basic tasks like the removal of background. It may also be used when the training data is sparse. In transfer learning, the theory is that a model which has been trained for one purpose can easily be adapted to suit a different use case by transferring the learned knowledge to the other challenge[8]. This is achieved by keeping the main structure and weights of the neural network and only changing the last layers to apply the way of classifying the final output. The lower layers, therefore, take care of more general tasks like identifying an object in an image[9] and the higher layers, the more specialised task. Not only does that enable the opportunity to achieve great results with only a small training dataset, but one also benefits from the big research field, which initially trains the model with more computational power. Transfer learning is especially applied in the field of computer vision, which is why it was chosen as a promising approach for the underlying research question.

For the detection of American SLA, we may use (to a degree) any pre-trained model for object detection on our dataset of American SLA signs even if they are limited in count and quality, as they only make the last layer of our final model.

To determine whether a model may be suitable for our application, we use all of the models and their variants available in Keras[7], as shown in Table 1. The *Avg. Top-5 Accuracy* in the table refers to the average accuracy of all variants combined.

In order to define, which of the above stated models we will further evaluate and possibly optimize, we train each of the Keras models in an experimental setup. The experimental training is defined by:

1. Training with 5% of the dataset (4.300 images, equally distributed to the 29 targets), with an 80/20 Train-Test-Split
2. 10 Epochs of Training
3. No Early Stopping

The results will focus on two key indicators: accuracy on our dataset and training time.

The following subsections will give a brief introduction of the most relevant model families and how they differ.

Table 1: *Keras Applications*

Name	Avg. Top-5 Accuracy	Total	Variants (*=Model)
Xception	0.945	1	*
VGG	0.901	2	*16, *19
ResNet	0.931	6	*50, *101, *152, *50V2, *101V2, *152V2
Inception	0.945	2	*V3, *ResNetV2
MobileNet	0.898	2	*, *V2
DenseNet	0.930	3	*121, *169, *201
NASNet	0.940	2	*Mobile, *Large
EfficientNet	-	8	*B0, *B1, *B2, *B3, *B4, *B5, *B6, *B7

4.3.1. VGG16

The VGG models are one of the earlier computer vision models and were introduced in 2015 by Simonyan and Zisserman who showed that using deep architectures with rather small filters can be superior to other models at that time[10]. VGG represents a classical convolutional neural network that takes 224x224x3 (width x height x channels) input pictures and passes it to multiple convolutional layers, pooling layers and activation functions to a fully connected final layer for classification. It includes in total 16 layers with weights and is shown in figure 2.

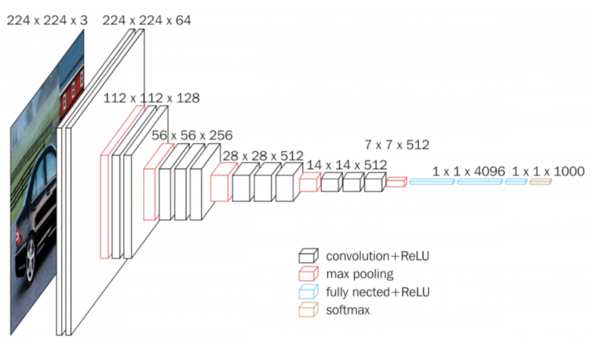


Figure 2: *VGG16 architecture[10]*

The tremendous change compared to previous convolutional neural networks is that the used filters and layers have the same size and parameters such as only ReLu as activation function[10]. This newly introduced simplicity paired with a relatively deep architecture led to unique results in the ILSVRC-2012**??** and ILSVRC-2013**??** competitions compared to the previous state-of-art AlexNet. On top of that is was shown that the model had a strong ability to generalise over multiple datasets and still achieve a top 5 performance. The authors of the network structured it in a way that it is possible to move from 16 layers to 19 layers for even better results.

4.3.2. ResNet50V2

Due to the fact that the performance of a very deep neural network started to decrease at some point, and the vanishing gradient while backpropagating became another problem, He et al. introduced the ”residual units” for the new architecture[11]. The main idea is to not only pass processed information from layer to layer but also to layers further ahead in the network. This skipping of e.g. one layer has been introduced as ”skip-connection” or ”identity shortcut.” The layers and their connections to other layers are arranged as residual blocks which share the same input and output size within the block[11]. The connection from one block to another is done by a ”projection shortcut,” which allows a change in input and output dimensions.

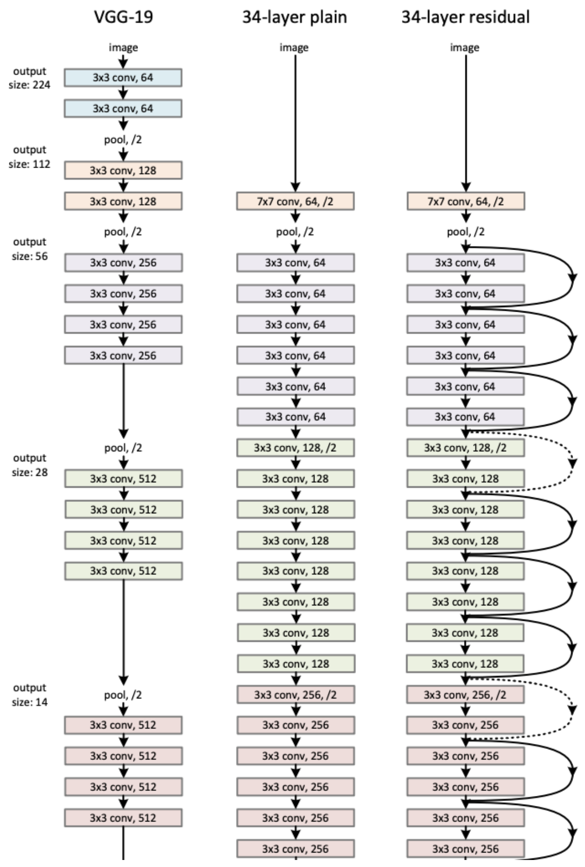


Figure 3: *ResNet architecture[11]*

Figure 3 presents the different residual units on the right side distinguished by the color. The dotted connections represent the projection shortcuts, whereas the solid arrows represent the identity shortcuts. This newly introduced architecture enabled the authors to solve both of the above mentioned issues and build deep, high performing neural networks. The main reason for this achievement is that higher layers can learn important information from the lower layers directly.

4.3.3. MobileNet

MobileNet was developed by Google Inc. with the purpose of being used for mobile and embedded vision applications. This target leads to the fact that the model has been optimised for latency. The authors introduced ”depthwise separable convolutions,” in which the channels are separately convoluted on their own and only then combined through a 1x1 pointwise convolution[12]. A visualisation can be found in figure 4. This separation of the convolution for each channel and a followed combination of the feature map results in significantly less computation compared to the previous neural networks and therefore minimizes the latency drastically by approx. 8 times[12].

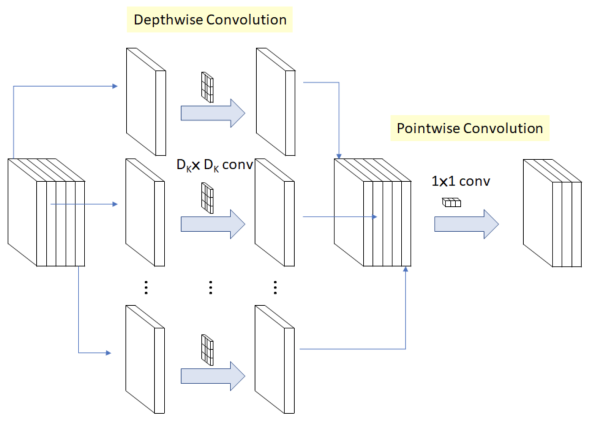


Figure 4: *MobileNet architecture[12]*

Besides the tremendous success with focus and latency such as high performance on the usual benchmark datasets for e.g. facial recognition, the authors found out that with this approach, less regularisation and data augmentation is needed to achieve state-of-the-art results.

4.3.4. MobileNetV2

The second version of the MobileNet introduces two additions to its predecessor. The first one is called the bottleneck layer, where a low-dimensional feature vector is passed to the respective layer, which then expands this vector to a high-dimensional space, applies a depthwise convolution and reduces the dimensions again to deliver the output[13]. This way, the number of computations through out the entire network is kept stable, which has a positive impact on the latency. As this technique leads to some loss of contained information, the authors introduced residual units which follow the same logic like the units explained in section 4.3.2. This way, information can be passed by skipping a layer and the information loss due to the bottleneck layer is reduced.

Table 2: *MobileNet comparison*

Version	MACs (millions)	Parameters (millions)
MobileNetV1	569	4.24
MobileNetV2	300	3.47

Table 3: *MobileNet FPS comparison*

Version	iPhone 7	iPhone X	iPad Pro 10.5
MobileNetV1	118	162	204
MobileNetV2	145	233	220

Additionally, the authors showed increased performance on benchmark datasets such as a potential combination with single shot detection (SSD), which can yield into even better and faster results.

4.3.5. DenseNet 201

Dense convolutional networks, also called DenseNet, were introduced in 2016 by Huang et al. and proposed a new way of building very deep neural networks which are stable to train and high performing. The change compared to other architectures was that each layer is connected to all following layers to pass the resulting feature vector through the entire network[14]. A visual illustration can be found in figure 5.

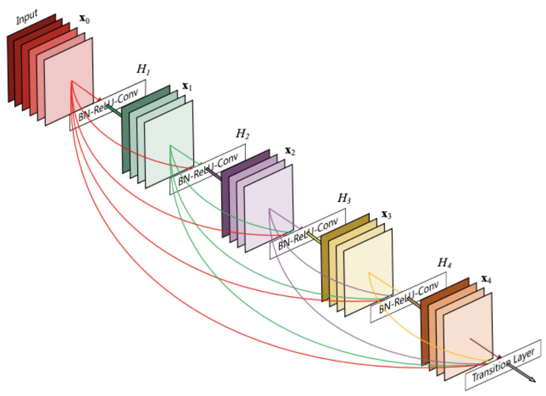


Figure 5: *DensNet architecture[14]*

The connection from one layer to all following layers is in a typical fast-forward manner which leads to the following advantages:

- alleviate the vanishing-gradient problem
- stronger feature re-use
- smaller number of parameters

The reduced number of parameters is a result of the network being able to ignore redundant feature maps which do not need to be learned again. In the initial paper, the authors propose an architecture with very narrow layers (12 filters per layer)[14]. Due to the complete connection of all layers, the model is less likely to overfit and more stable to train. The model contains residual units as well, which shall strengthen the information flow from previous to later layers. Additionally, bottleneck layers can be found, which were introduced to minimise the number of computations being caused by the complex connectivity.

4.3.6. Xception

Xception is an advanced version of Google's Inception model and a short form of "Extreme Inception". The previous architecture of Inception convoluted the input channel by channel and combined the results in the end. This depthwise separable convolution was slightly modified in the newly introduced Xception model by Chollet[15]. The author changed the regular setup of first applying a depthwise convolution and then the pointwise to a setup that executes these steps the other way round. However, this change on its own did not lead to major improvements with regard to performance which is why a second change was applied by introducing residual connections[15].

Figure 6 shows a clear improvement of the introduction of residual blocks.

Furthermore, the author removed the ReLu activation after the depthwise convolution, different than in the Inception

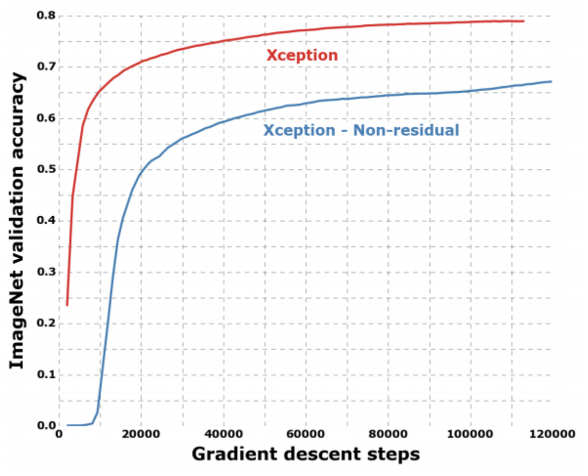


Figure 6: *Impact of residual units for Xception[15]*

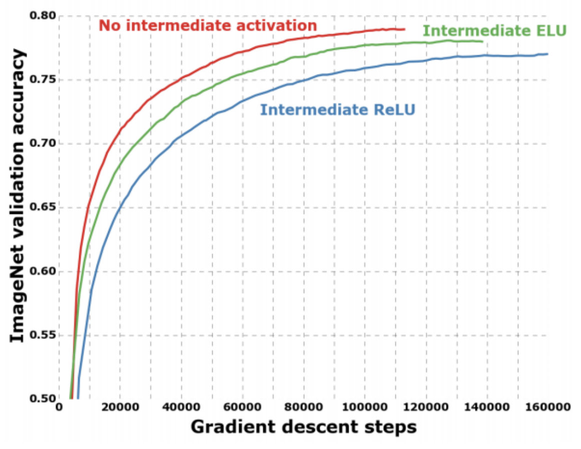


Figure 7: *Impact of removing ReLu activation[15]*

architecture, which had a tremendous impact on the model performance.

The changes not only led to a similar model size like Inception-V3 but also outperformed VGGNet, ResNet, and Inception-V3 in accuracy with regard to known benchmark datasets[15].

4.4. Evaluation metric

The following evaluation will use the model accuracy as a final metric to define each model's performance. The accuracy will be calculated for each target variable and then averaged over all classes. The calculation can be described as:

$$ACC = \frac{1}{N} \sum_{i=1}^{N_c} \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \qquad (1)$$

$N_c = \textit{Number of classes}$
$TP_i = \textit{Count of true positive predictions for given class}$
$TN_i = \textit{Count of true negative predictions for given class}$
$FP_i = \textit{Count of false positive predictions for given class}$
$FN_i = \textit{Count of false negative predictions for given class}$

5. Results

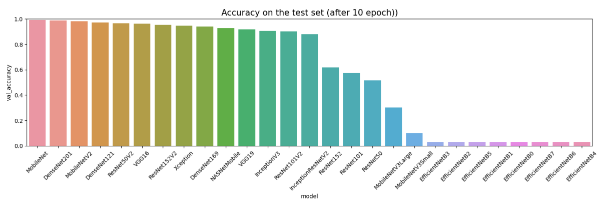
In order to find the most suitable algorithm for translating a live input stream of American SLA to any other suitable SLA (sharing the same alphabet) in a real-world application, the algorithm needs to excel in two main aspects: accuracy and inference time. A third factor that may come into question here is the training time, as the computational cost may accumulate when further improving the model in the future.

5.1. The long list

As mentioned in chapter 4.3 the long list contained 27 pretrained models. Due to the limited availability of computational resources, the models on the list as per table 1 (including all variants) were trained in an experimental setup to determine which models are suitable for full training. The goal was to use these pretrained models and only train the last layer to enable the detection of SLA in sources such as webcam feeds, videos or images.

The experimental setup was trained on a consumer pc, with a 3.8 Ghz 6-core CPU, a *GeForce GTX 1060* GPU and 32GB of RAM. The results mainly focus on two aspects in order to decide which models may be suitable for the full training: training time and accuracy. As shown in Fig. 8 some of the models already have high accuracy with a minified training.

Figure 8: *Accuracy on long list training*



While models as *DenseNet* seem to have good results with all variants, others like *ResNet* have a high variety; accuracy ranges from 55% (*ResNet50*) to 98% *ResNet50V2*. The *EfficientNet* with all its variants has an accuracy below 10% and will not move forward.

Figure 9: *Training time on long list training*

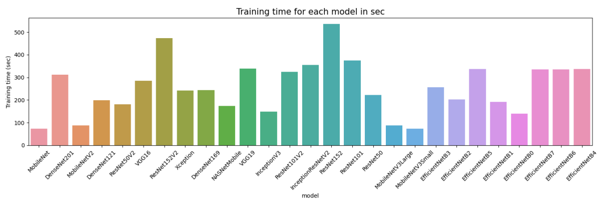


Fig. 9 shows the time that was needed to train the models with the reduced dataset for 10 epochs without early stopping. While it shows a wide range, the times train not too long at all. Therefore, the training time will not be the main factor in deciding which model we should move forward with.

When deploying the model in a real-time environment, such as live detection via a webcam, another factor comes into play: the inference time. As described in chapter 4.3, the models differ fundamentally in their architecture. At this point, it is uncertain how the architectural structure will affect the inference time of a fully trained model. Therefore we decided to move forward with the best performing model from each architecture described in chapters 4.3.1 to 4.3.6. In addition to the top 5 models, we decided to add *MobileNetV2* as well, as it seemed interesting why the newer version of the model did perform worse than the old one. Table 4 contains all models that will be used for full training.

Table 4: *Short List candidates*

Name	Acc.	Time (seconds)
MobileNet	0.9957	82
MobileNetV2	0.9756	165
DenseNet201	0.9842	328
ResNet50V2	0.9239	178
Xception	0.9698	238
VGG16	0.9382	286

5.2. The short list

For the short list produced in chapter 5.1 the training was slightly adapted. In contrast to the training of the long list, Early Stopping with a *patience=1* was implemented in order to stop the training after 1 epoch without any improvement of the loss rate to avoid an overfit. Since no further knowledge for expected epochs needed was given, a maximum of 50 epochs was configured. Table 5 shows the result of the training.

Table 5: *Short List Results*

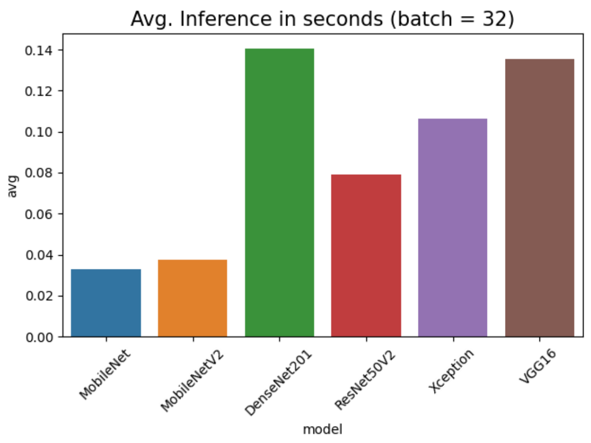
Name	Acc.	Change %	Time (seconds)	stopped at
MobileNet	0.9921	-0.36	160	2
MobileNetV2	0.9900	+1.48	385	4
DenseNet201	0.9905	+0.64	1058	3
ResNet50V2	0.9840	+6.51	578	3
Xception	0.9820	+1.26	746	3
VGG16	0.9960	+6.16	1975	7

As we can see, the results did not significantly change in comparison to the minified setup, reflected by the *Change %* column. Although using the full dataset instead of only 10%, the training time did not increase the expected folds due to the early stopped training. As in the minified training, *MobileNetV2*'s accuracy remains lower than its predecessor *MobileNet*, even with a later stopped training. At this point, it still remains unclear why this can be observed.

5.3. Inference Time

As mentioned above, training time and accuracy are not the only factors when choosing a suitable model for a real-time application. Accuracy is high across all fully trained models, and while training time differs within the short list results, it can be disregarded as the total times are relatively short, even when the training runs on a consumer machine. Training time also does not accumulate due to the early stopping mechanism. In order to select the best model, the inference time is the crucial value as it determines how many frames (predictions) the model can handle per second, a key factor for a real-time application. The inference time was measured with the above saved models. All models executed the predictions on the same initially randomly selected 544 images with a batch size of $n = 32$. The process was repeated 10 times with different randomly selected images.

Figure 10: *Avg. Inference in Seconds*



The results visualized in Fig. 10 show that only *MobileNet* and *MobileNetV2* have near real-time capabilities. While a prediction time of 0.14 seconds (*DenseNet201*, Fig. 10) does seem acceptable to the human eye, it translates to approx. 7 predictions per second. Given that the human eye can detect meaning in images within 13ms[16] (approx. 75 frames per second (fps)) and videos have a minimum framerate of 23.97 fps to appear smooth and fluent, 7 predictions (frames) per second will not lead to an acceptable user experience. Therefore only the *MobileNet* and *MobileNetV2* should be considered for real-time applications, as the inference time of 0.03 seconds leads to a framerate of approx. 33 fps.

5.4. Shipping production-ready Model

The final implementation of the trained model is up to the user. Each trained model is saved as a portable *Tensorflow* file and can be used to implement a real-time detection via *OpenCV*, for example. In order to confirm the usability of the model, we implemented an application, which takes static images as an input and outputs them according to the image of a predefined SLA dataset of another language.

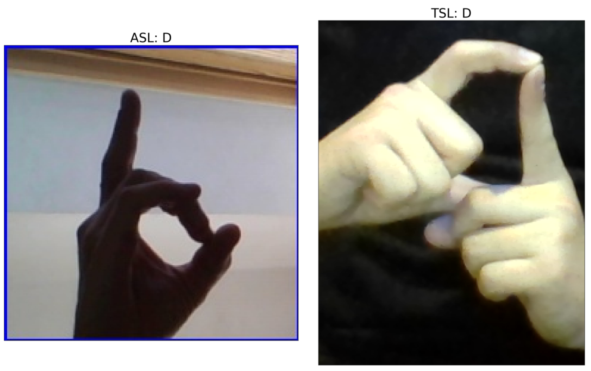
The target SLA for the sample implementation was the Turkish Sign Language (TSL). Against our expectations, the *MobileNetV2* did significantly worse than *MobileNet*. Tables 2 and 3 suggested a faster inference time, but our research showed, that *MobileNetV2* was 33% slower than *MobileNet*. Therefore the *MobileNet* should be used for a production environment and is used in our sample implementation.

As any real-time application would have, the dimensions and resolution of the input data differed from the images used in training. Fig. 11 shows a sample output for the translation of the left sign (ASL Letter D), the predicted sign and the according to output image for the TSL.

6. Discussion

The training of the long and the short list in chapter 5 diminished the importance of the training time, but therefore the key factor for choosing the best model for the application became the inference time. *MobileNet* has the best inference time of 0.03 seconds per prediction, allowing a framerate of approx. 33

Figure 11: *Sample Translation from ASL to TSL, Letter D*



fps. While this can result in a smooth video in real-time applications, many of the current webcams allow a higher framerate of up to 60 fps. For the model not to slow down the actual output, the next steps should focus on the improvement of the inference time.

We identified three major leverage factors for improving the inference time: reducing the input layers, adjust the training settings in *Keras* and implement a distributed prediction with *DKeras*.

6.1. Reducing the input layers

Improving the inference time of our model can be achieved by a process called *Knowledge Distillation*[17]. Models typically use a *softmax* output layer to convert the logarithmic value of the model to a probability by comparing it to the other logarithmic values. Geoffrey Hinton et al[17] propose a method to derive a distilled model from the original one by averaging two weighted functions. The first function uses the cross entropy with the soft targets; the second function is the cross entropy with the correct labels. Tests with various datasets showed that the speed of the distilled models increased while the accuracy decreased by 1-5%, depending on the dataset and architecture.

The accuracy of *MobilNet* is at 99% (see Table 5), even with a decreased accuracy of approx. 94% with a distilled model, assuming an unchanged inference time of 0.03s, this leads to a wrong label every other 2.82 seconds for one frame, down from 2.97s with a 99% accuracy. Since the actual prediction needs to be determined over a certain amount of frame, this is a non-factor. Even if the inference time does not increase by a significant margin, any improvement can lead to a better user experience and therefore should be tested.

6.2. Adjust the precision

Another possible improvement of the inference time is to decrease the precision of the weights. This can be achieved in two ways: pre or post training.

The *Post Training Quantization*[18], a method implemented in *Tensorflow*, converts the weights after the training from floating points of 32-bits to integers of 8-bits. This decreases the accuracy of the model, but therefore significantly improves the model size in memory, which leads to a much lower latency of the model.

Between the 32-bits and 8-bits precision, there is also the option to convert the weights to a half-precision approach of 16-bit floating points. The *Mixed Precision Training* reduces the model size by 75% but needs to be specified before the training. During training, the 16-bit weights may result in an underflow of the weight (the value becomes zero due to the gradient being too small for the half-precision), causing the model to learn incorrectly. Not all layers play the same role with the inference time. The *Automatic Mixed Precision* enables the half-precision only for optimizable layers, such as the convolutional layers.[19]

Both approaches may be suitable for further testing, as in chapter 6.1 mentioned before, a small loss in accuracy won't be noticeable for the user if implemented correctly.

6.3. Implement dKeras

Distributed Keras Engine (dKeras)[20] is an open-source project that focuses solely on improving the inference time of models used in production environments without the need for special hardware. The core of the project is to run the predictions in parallel and therefore improve the inference time significantly without needing to re-train or convert the used model. The documentation shows that only two lines of code need to be replaced in order to implement the optimized predict method. According to the benchmark and depending on the architecture of the used model and the available dKeras workers, the inference time improves approx. 94-97% without decreasing the accuracy.

Using dKeras for a fully trained model could decrease the importance of the inference for all models. A 94% improved inference time for the *DenseNet201* (the slowest of the models, see Fig. 10) leads to an inference time of approx. 0.0084 seconds, a framerate of 2738 fps. At this point, no consumer hardware has the capability of outputting that framerate.

Further improvements should focus on the implementation of dKeras.

7. Conclusions

This scientific work proposed an applied review of multiple state-of-art deep learning models to elaborate their usability for sign language translation. The specified problem of not having intense research in the field of translating letters from one sign language to another was the main motivation. The goal was to compare multiple models with regard to accuracy such as latency to make an educated guess of which of the proposed models is the most suited one for a user-facing application. To validate which model to choose, an 87.000-image big dataset with American sign language letters was used for the training. Starting with a long list of 27 pre-trained models from the Keras framework, a short list of only the 5+1 best performing models was chosen to proceed with. The models were trained by applying transfer learning to maintain the previously learned weights. All models were compared based on their accuracy and prediction time such as the number of frames per second which they can process. It was shown that the accuracy of all models was quite similar, which led to the fact that no model could have been proposed solely based on this indicator. However, due to the different model architectures, the research showed a significant difference in the prediction time such as the number per frames which were processed. *MobileNet* resulted as a preferred model from this comparison as the model accuracy such as its prediction speed were superior compared to the other architectures. Nevertheless, the current workflow is only applicable for languages that share the same letters as the predicted output is only mapped to the other language's letter. Further work has to include a more sophisticated word and letter embedding for the actual translation after a correct prediction. For improving the inference time of any given model, the implementation of *dKeras* should be considered in a real-time productive environment. Also, the actual performance of an implemented infrastructure has to be tested as all experiments were executed in-vitro.

8. References

[1] WHO. (2021) Who: 1 in 4 people projected to have hearing problems by 2050. [Online]. Available: https://www.who.int/news/item/02-03-2021-who-1-in-4-people-projected-to-have-hearing-problems-by-2050

[2] A. Wadhawan and P. Kumar, "Sign language recognition systems: A decade systematic literature review," *Archives of Computational Methods in Engineering*, vol. 28, no. 3, pp. 785–813, 2021.

[3] M. J. Cheok, Z. Omar, and M. Jaward, "A review of hand gesture and sign language recognition techniques," *International Journal of Machine Learning and Cybernetics*, vol. 10, pp. 131–153, 2019.

[4] C. Dong, M. C. Leu, and Z. Yin, "American sign language alphabet recognition using microsoft kinect," in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015, pp. 44–52.

[5] A. Joshi, H. Sierra, and E. Arzuaga, "American sign language translation using edge detection and cross correlation," in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2017, pp. 1–6.

[6] M. Taskiran, M. Killioglu, and N. Kahraman, "A real-time system for recognition of american sign language by using deep learning," in *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, 2018, pp. 1–5.

[7] F. Chollet *et al.* (2021) Keras applications. [Online]. Available: https://keras.io/api/applications/

[8] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," 2020.

[9] B. Neyshabur, H. Sedghi, and C. Zhang, "What is being transferred in transfer learning?" 2021.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[14] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[15] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017.

[16] M. C. Potter, B. Wyble, C. E. Hagmann, and E. S. McCourt, "Detecting meaning in rsvp at 13 ms per picture," *Attention, Perception, & Psychophysics*, vol. 76, no. 2, pp. 270–279, Feb 2014. [Online]. Available: https://doi.org/10.3758/s13414-013-0605-z

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[18] Tensorflow. (2021) Post training quantization. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/quantization/quantize

[19] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2018.

[20] dKeras. (2021) Distributed keras engine. [Online]. Available: https://github.com/dkeras-project/dkeras