Research topic:

A comparison study on detection of Deepfakes using ANNs

Subject: Computer Science

Word count: 3988

Table of Contents

Introduction

Over the past decade technology has developed at a revolutionary pace, and with it many challenges. Among these, "deepfakes" is one of the most prominent looming threats of 2020 [1]. Deepfakes are "hyper-realistic videos that apply artificial intelligence (AI) to depict someone say and do things that never happened" [2]. This technology has existed for decades and while it used to take entire studios full of experts to realistically manipulate one video, the process has been democratized because of novel machine learning (ML) and deep learning (DL) techniques [3]. As the quality of deepfakes increase and become harder to identify as synthetic, building reliable automatic detection strategies becomes vital.

Currently, according to computer-science professor and digital-forensics expert at the University of California at Berkeley Hany Farid, "The number of people working on the video-synthesis side, as opposed to the detector side, is 100 to 1" [4]. This unbalance is alarming, and the effects can be devastating. In a report by Robert Chesney and Danielle Keats Citron they outlined the extent of the potential danger of deepfakes [5]. They wrote that "[because of deep fakes] individuals and businesses will face novel forms of exploitation, intimidation, and personal sabotage. The risks to our democracy and to national security are profound as well."

I have chosen to compare multiple deep learning models in the process of detecting and classifying deepfakes. The motivation for this essay is to contribute to the efforts of limiting the influence of deepfakes. Specifically, the models are Convolutional Neural Networks (CNN), two Recurrent Neural Networks (RNNs) variations – including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) – and Multi-Layer Perceptrons (MLPs). For each model, I will test varying configurations of parameters in the models. This is important because it will illuminate which architecture provides the most promising results in the task of detecting deepfakes and I will compare the networks based on the training metrics, training speed, size (number of parameters and model file size) and rate of learning. I aim to come to a reasoned conclusion on which DL architecture is the most suitable for detecting deepfakes.

The paper will be organized into three sections. The materials and methods section will include background theory, dataset specifications, hardware requirements and the method of the experiment. The results section will include findings from the experiment. Lastly, the discussion section will analyze the results and discuss significant findings, in addition to evaluating and comparing the models.

1 – Materials and methods

1.1 Theory

1.1.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are engineered models of the human brain. ANNs are constructed with artificial neurons and axons (see left image in figure 1 below), where the neuron performs the computational steps and the axons form weighted connections between layers of neurons.

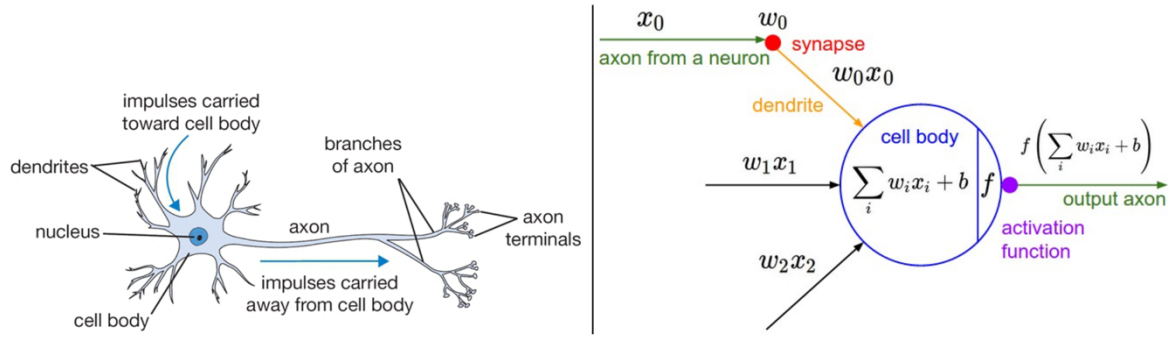

Figure 1: A comparison of the biological braincell and an artificial neuron. A computation of the neuron includes summing the inputs and using an activation function to determine the output value. Source: [6]

The output can thus be expressed mathematically as

$$output = f\left(\sum_{i=0}^{n} w_i x_i + b\right)$$

Where $w_i x_i$ is the ith weighted input, b is the bias, n is the number of inputs and f is the activation function. Bias is a constant added for increased learning capabilities [7]. Together, weights and biases make up the parameters of the model.

The depth is the number of layers and the width is the number of units in each layer (see figure 2), two variables used in the experiment described in later sections.

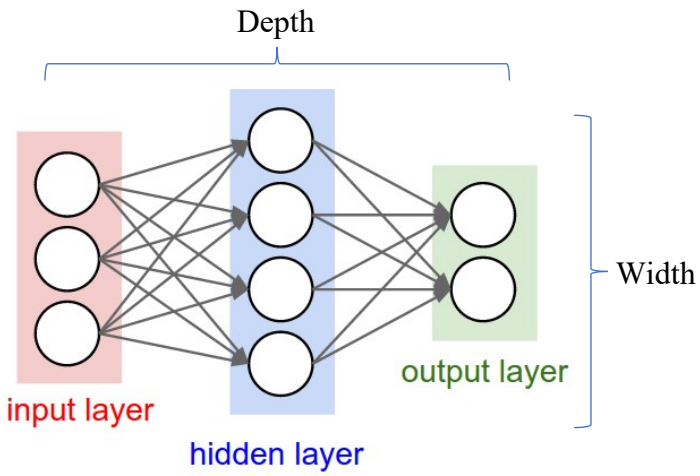

Figure 2: A simple 3-layer ANN, with nodes and connections illustrated. Source: [6]

For the neural network to be effective, the parameters must be tuned/optimized. This is done by comparing the output of the network with the truth and computing a loss function (also referred to as error or cost), and then adjusting the parameters to minimize the loss. While this becomes analytically impossible for larger networks, gradient descent is a numerical method that addresses such a problem [8]. The goal of adjusting the parameters is to reach the global minimum of the loss function.

In the training process, the parameters are initially randomly generated resulting in poor starting performance. However, several iterations of training with a subset of the data – and updating the parameters after each iteration using a backpropagation algorithm to minimize the loss – results in a model with better performance. The model is tested on unseen validation data after every iteration to track its performance.

1.1.2 Deep neural networks

A Deep Neural Network (DNNs) is a subfamily of ANNs. The defining characteristic of DNNs is their depth. According to Eda Kavlakoglu (program manager at IBM) "A neural network that consists of more than three layers—which would be inclusive of the inputs and the output—can be considered a deep learning algorithm" [9].

Deeper ANNs "allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction" [10]. Therefore, DNNs are commonly used for more complicated and difficult tasks, and specifically an application relevant for this essay – video and image classification [11].

1.1.3 Convolutional neural networks

Convolutional Neural Networks (CNNs) are deep learning models consisting of special layers that perform feature scaling by extracting the most important details in a data matrix (e.g. image pixels) [12]. Feature scaling of images is the process of extracting specific relevant patterns – key features, such as edges, thus removing unnecessary information, which makes it much faster and computationally efficient to accurately classify images [13]. This process of feature extraction is made possible by two specific types of layers:

The *convolutional layer* is often the first layer used in a CNN and has a kernel size, stride and filter number (see figure 3). In the process of extracting the feature map, "we overlap the convolution kernel on top of the input image, [thus] we can compute the product between the numbers at the same location in the kernel and the input, and we get a single number by summing these products together" [14].

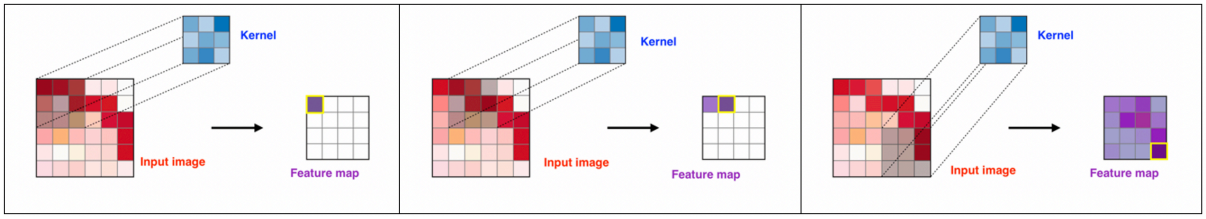

Figure 3: Illustrating the convolutional operation. The kernel is represented by the blue 3x3 matrix and the input image as the red 6x6 matrix. The stride (how far the kernel is moved between each convolution operation) is 1, as seen in the left and center image. ***Left:*** *first step.* ***Middle:*** *second step.* ***Right:*** *last step, finalizing the feature map. Source: [15]*

Moreover, this convolutional sequence is done with many unique kernels to extract different features (see figure 4). In this experiment, the different CNN models will vary the number of filters used, investigating the impact on performance.

(a) Lenna (b) Horizontal edge (c) Vertical edge

Figure 4: Example of how different filters capture different features of the same image. ***Left:*** *original image.* ***Middle:*** *horizontal edge feature map.* ***Right:*** *vertical edge feature map. Source: [14]*

The *pooling layer* is another type of layer commonly used in CNNs (see figure 5). Much like the convolutional layer, the pooling layer reduces the spatial size of the data while minimally losing critical features [16].

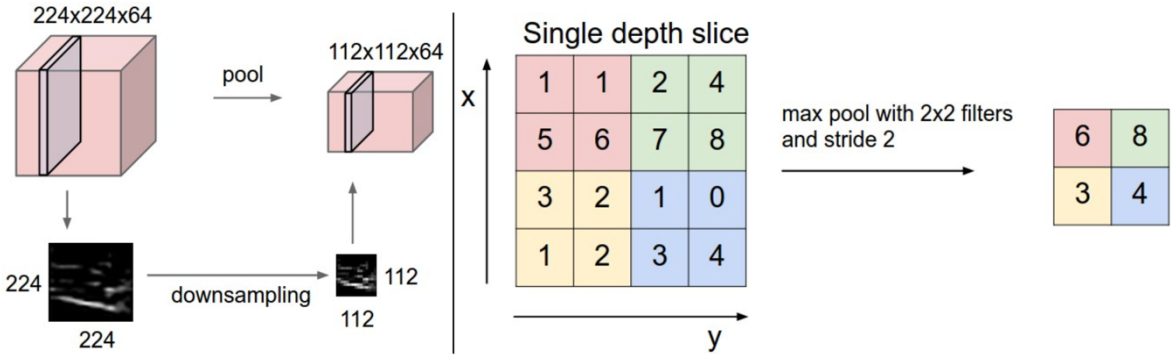

Figure 5: The pooling operation. ***Left:*** *Illustrates how every depth level is individually downscaled. The depth dimensionality remains the same between the input and output.* ***Right:*** *Shows a 4x4 matrix downscaling to 2x2. The filters/size determines how many values of the input matrix to compare, the max pool operation takes the maximum value in the selected matrix and stores that value. As shown in right image, the result is the smaller matrix. Source: [16]*

Another important component is the activation function between the convolutional and the pooling layers. "[T]he most successful and widely popular of these is the rectified linear unit (ReLU) activation", according to Auror, Raman et al. [17]. This ReLU activation function $\sigma(x)$ is given mathematically by:

$$\sigma(x) = \max(0, x) = \begin{cases} x, & if\ x \geq 0 \\ 0, & if\ x < 0 \end{cases}$$

This has become the standard activation function for CNNs, where x is the input of the node.

1.1.4 Recurrent neural networks

A major limitation of the previously mentioned neural networks is their temporal unawareness. For example, in video classification CNNs can only process one frame at a time, while lacking the context of previous frames. Recurrent neural networks (RNNs) are another form of ANNs which solve this problem by having feedback/feedforward loops, allowing information and data to persist within the network [18] (see figure 6 and 7). Therefore, the RNNs are temporally aware with contextual understanding and specifically used for processing sequential and time series data [19]. In addition, most RNNs can process sequences of variable length [20].

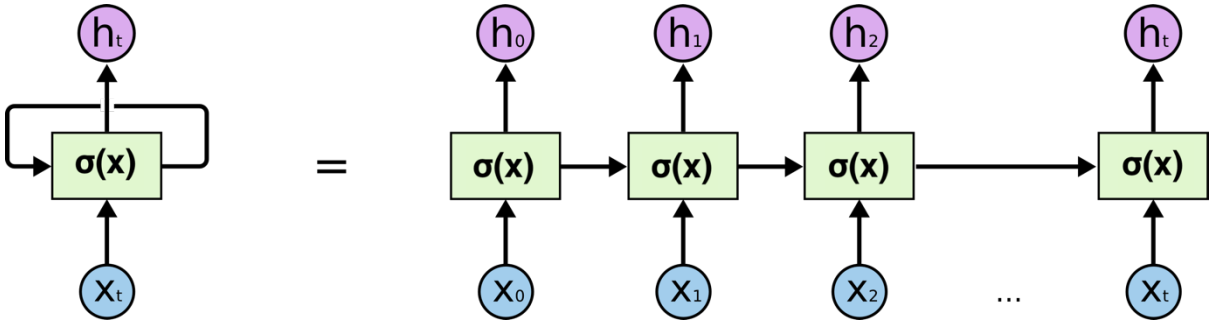

Figure 6: Architecture of a simple RNN, illustrated in two ways. ***Left:*** *Loop variant. While it may seem like this version only has one recurrent unit, one can imagine all the units in the layer being behind the first.* ***Right:*** *Un-rolled variant, where each unit is represented separately. Source: [18]*

A limitation of RNNs is the vanishing gradient problem. According to Amidi, Afshine et al. [21], "The reason why [the vanishing gradient] happen[s] is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers." Learning patterns in long term data is therefore more difficult to capture.

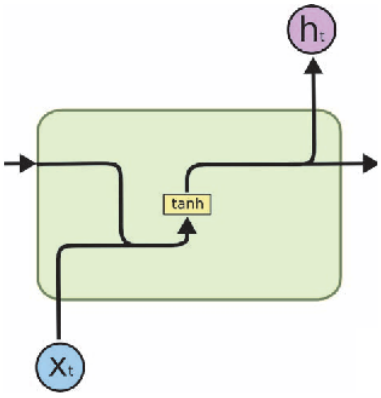

Figure 7: A simple RNN cell. The input at first timestep (x_0) is fed into the network. The output of the activation function $h_0=\sigma(x_0)$ is then the input of the next step in addition to the next data input x_1. This continues for all time steps t of the input sequence. Source: [18]

Two approaches to solving the vanishing gradient problem are Long Short-Term Memory (LSTMs) – first introduced by Sepp Hochreiter in 1997 [22] – and Gated Recurrent Units (GRUs) – introduced by Kyuonghyun Cho in 2014 [23]. These RNNs have different computational operations in the nodes, consisting of gates – controlling how data flows, is

stored for the next timestep or forgotten in the cell state. LSTMs and GRUs use a similar but somewhat different structure (see figure 8).

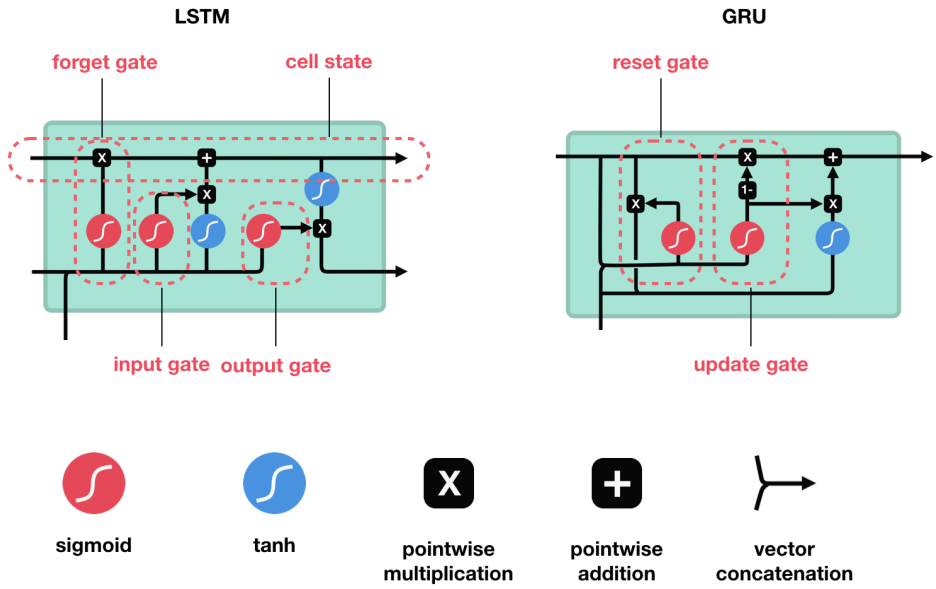

Figure 8: Comparison of the gates of LSTM and GRU cells. ***Left:*** *LSTM cell, showing 3 gates. The forget gate discards information from the previous time step, the input gate stores information in the current time step and the output gate passes information from the cell state to the next time step [18].* ***Right:*** *GRU cell, showing 2 gates. The update gate passes information from the previous time step to the next step and the reset gate forgets information from the previous step [25].* ***Bottom:*** *Description of symbols and operations within the cells. Source: [24]*

1.2 Materials

1.2.1 Hardware

The experiment was run on a desktop Windows computer with a GeForce GTX 1060 6GB GPU (CUDA enabled), an Intel i7-7700 3.6 GHz Quad-core CPU, 24 GB DDR4 RAM and 512 GB M.2 SSD. For training the models, Keras was used with a TensorFlow 2.3.0 backend, and CUDA 10.1 was installed (for GPU acceleration) [26].

1.2.2 Dataset

The FaceForensics++ dataset [27] consists of both manipulated and real videos from sources such as YouTube and scenarios with paid actors. The deepfakes were created with 4 different manipulation techniques (DeepFakes, Face2Face, FaceSwap and NeuralTextures). The Deep Fake Detection dataset by Google and Jigsaw [28] was also used and these datasets combined in total make up 2720 manipulated and real videos.

As done in the FaceForensics++ paper [27], 1000 full resolution videos will be used in total, with an equal amount of fake and real videos and durations ranging between 5 seconds and 2 minutes. An 80% training and 20% testing data split is used. For classification of deepfakes, individual frames must be extracted from the videos. Regardless of the video length, 40 frames evenly spaced throughout the video was chosen (see figure 9). Lastly, the images were cropped to only include the face of the subject (see figure 10).

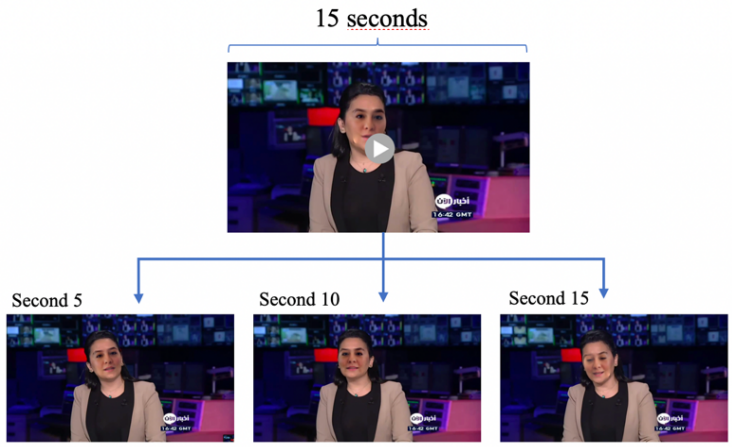

Figure 9: Splitting the videos into frames. For the sake of simplicity, this only shows 3 frames. The experiment uses 40 frames from each video. Source: video from the FaceForensics++ dataset [27].

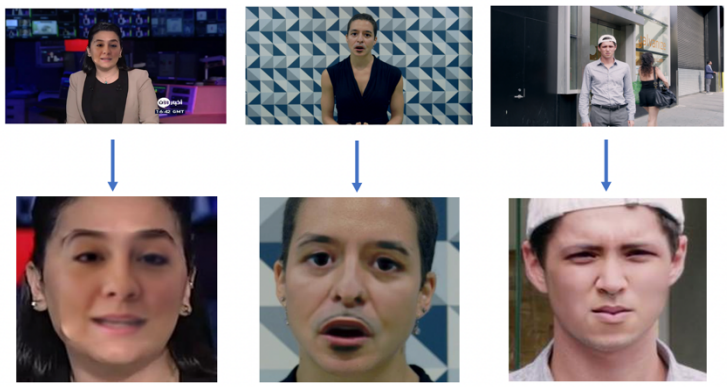

Figure 10: Examples of cropping the frames to focus on the subject's face using the "Face Recognition" Python library [29]. Source: FaceForensics++ dataset [27].

1.3 Experimental Method

In this essay, tests will be conducted to compare the custom CNN variations, transfer learning with a pretrained CNN model, RNN variations and MLPs. All models will be trained for 60 iterations. Every test will be done 10 times, and the average results will be presented with a standard deviation. This is to reduce effect of random weight initialization (which can lead to varying and inaccurate results). Figure 11 illustrates the experiment outline, visualizing the structure of various test combinations.

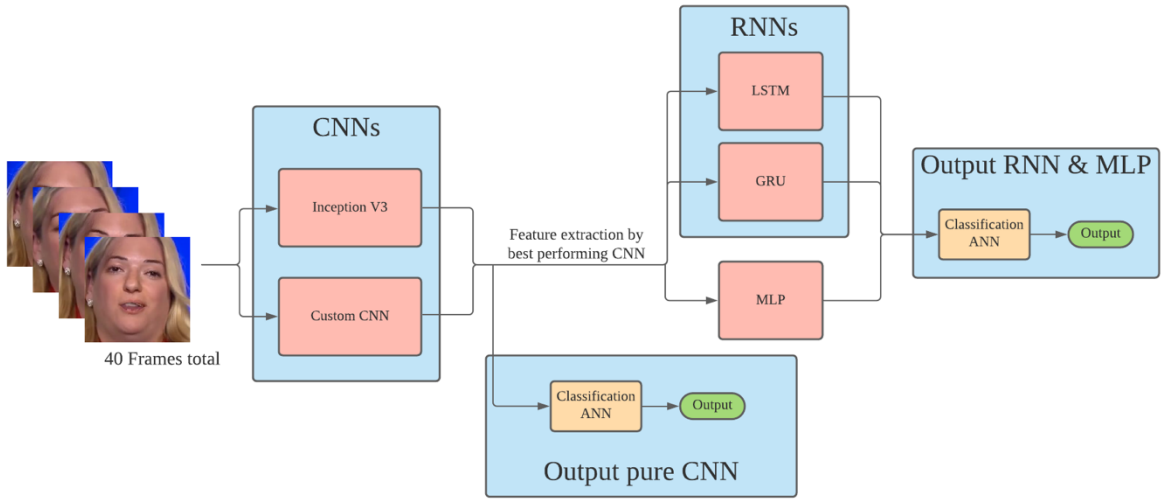

Figure 11: Experiment outline. 40 frames/faces (represented by the images on the left) are inputted into the CNNs. One part of the experiment classifies these features with an ANN and gets that output (bottom green box). However, the extracted features from the best performing CNN are also used as inputs in the RNNs and MLP. The output of these are then classified with another ANN (right green box) which determines if the frames were real or deepfakes. Red boxes show different DNN models, blue boxes indicate the type of model, and yellow boxes show the simple classification ANNs. Source: author.

1.3.1 Custom CNNs

For the custom CNNs, 4 depth levels (2-5 layers) will be used with varying width configurations. In table 2, the number of filters of individual layers in each depth level of the CNNs is shown.

2-layer model		
Depth / Width	Layer 1	Layer 2
Ascending	64	128
Descending	256	64
Constant 32	32	32
Constant 64	64	64
Constant 128	128	128
Constant 256	256	256

3-layer model			
Depth / Width	Layer 1	Layer 2	Layer 3
Ascending	64	96	128
Descending	256	64	32
Constant 32	32	32	32
Constant 64	64	64	64
Constant 128	128	128	128
Constant 256	256	256	256

4-layer model				
Depth / Width	Layer 1	Layer 2	Layer 3	Layer 4
Ascending	32	64	96	128
Descending	256	128	64	32
Constant 32	32	32	32	32
Constant 64	64	64	64	64
Constant 128	128	128	128	128
Constant 256	256	256	256	256

5-layer model					
Depth / Width	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
Ascending	32	64	96	128	160
Descending	256	192	128	64	32
Constant 32	32	32	32	32	32
Constant 64	64	64	64	64	64
Constant 128	128	128	128	128	128

Table 1: Configurations of the custom CNN variations in regard to depth (layers) and width (number of filters). "Ascending" models will have increasing filter widths, "descending" models will have decreasing filter widths and "constant" models do not vary width. The 2-, 3-, 4- and 5- layer tables are all independent separate tables but have been grouped together for the sake of simplicity.

1.3.2 Inception CNN

Transfer learning "allows you to retrain the final layer of an existing model, resulting in a significant decrease in not only training time, but also the size of the dataset required" [30]. This approach is very effective because it allows the adaptation of state-of-the-art model architectures for use in another specific application – in this case, detecting deepfakes. The

model InceptionV3 by Google architecture [31] will be used – runner-up in the ImageNet Large Visual Recognition Challenge (trained on 1 million images).

InceptionV3 has 314 layers in total (see figure 12), and certain numbers of layers will be retrained and tested to understand which configuration is optimal. The model will retrain the:

1. Top 50% layers
2. Top 25% layers
3. Last 2 layers, which is the fully connected ANN at the top of the network

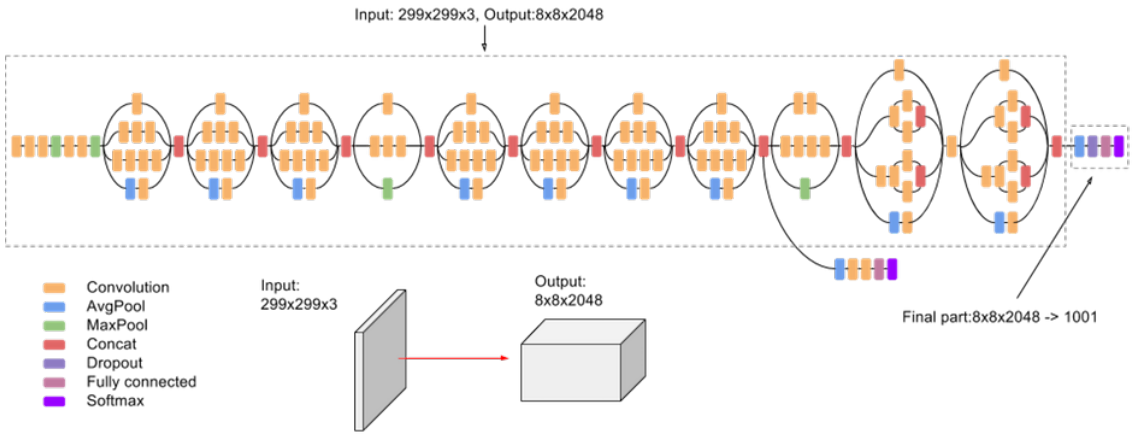

Figure 12: The architecture of the InceptionV3 model visualized. This architecture extracts features from a 299x299 RGB input image using a combination of the layers listed in the bottom left corner. This extraction output (8x8x2048 matrix) is inputted into the "final part" which is the fully connected layers at the right of the image. This last part outputs the prediction of the entire model. Source: [32]

1.3.3 Feature extraction

The next step is to train the LSTM, GRU and MLP models on a set of features from the of 40 frames from each video. To get the extracted features from a frame, the best performing CNN model can be used by removing the top ANN (fully connected layers) of the CNN. The features are sequential series data used as inputs in the RNNs and MLPs.

1.3.4 LSTM and GRU

With both LSTMs and GRUs, 4 depths will be tested, ranging from 1 to 4 layers. For each of these depths, 4 widths variations will also be tested; 128, 512, 1024 and 2048 units wide. For this part of the experiment, the units will remain constant between layers in each

configuration. In addition, the trained RNN models are combined with a 2-layer ANN with 1024 and 2 node units in the respective layers, to get a classification prediction.

1.3.5 MLP

Lastly, the MLP will also be tested to understand if the temporal awareness can also be learnt with feed forward ANNs. Table 2 shows the variations between depth and width. Each model also has a 2-unit top layer for the output of the classification result (not included in the table).

MLP Configuration	1 layer		2 layer		3 layer		
Depth / Width	1	-	1	2	1	2	3
Constant 128	128	Ascending	512	1024	64	160	1024
Constant 256	256	Descending	1024	512	1024	160	64
Constant 512	512	Constant 256	256	256	64	64	64
Constant 1024	1024	Constant 512	512	512	128	128	128
Constant 2048	2048	Constant 1024	1024	1024	256	256	256

Table 2: MLP model configurations describing number of layers (depth) and nodes (width). The gray columns describe the width of the model. "Ascending", "descending" and "constant" refers to how the width changes between layers.

1.4 Performance metrics

Making model success measurable and justifiable is essential for reliable conclusions. The results of the models will be evaluated based on several performance metrics.

Accuracy (A_t) and validation accuracy (A_v)

The chance that the model will make a successful prediction. Comparing A_t and A_v can give a good indication of potential overfitting. Both values range from 0 to 1, where 0 is worst and 1 is best.

Loss (L_t) and validation loss (L_v)

A measure of how bad a model's prediction is compared to the truth. As with accuracy, loss and validation loss reflect the overfitting of models. Lower values of loss are desirable. Both loss values range from 0 to ∞, where 0 is best and higher values indicates a worse performing model.

Training speed (T_s)

The time spent learning (from when training starts to when it stops). Each model must be trained for the same number of epochs (60) for comparable results. Training speed is measured in seconds.

Learning rate (L_r)

How fast the model converges on optimal performance metrics. A quicker convergence means the model needs less data to learn patterns. This is measured by examining which epoch the optimal metric value is achieved by the model.

Number of parameters (N_p) and weight file size (W_s)

The number of parameters in the model, in addition to the size in bytes of the saved weights file. This is important if the model needs to run on hardware with limited storage.

2 – Results

2.1 Custom CNN

Figure 13, 14 and 15 shows the data from the training of the custom CNN models.

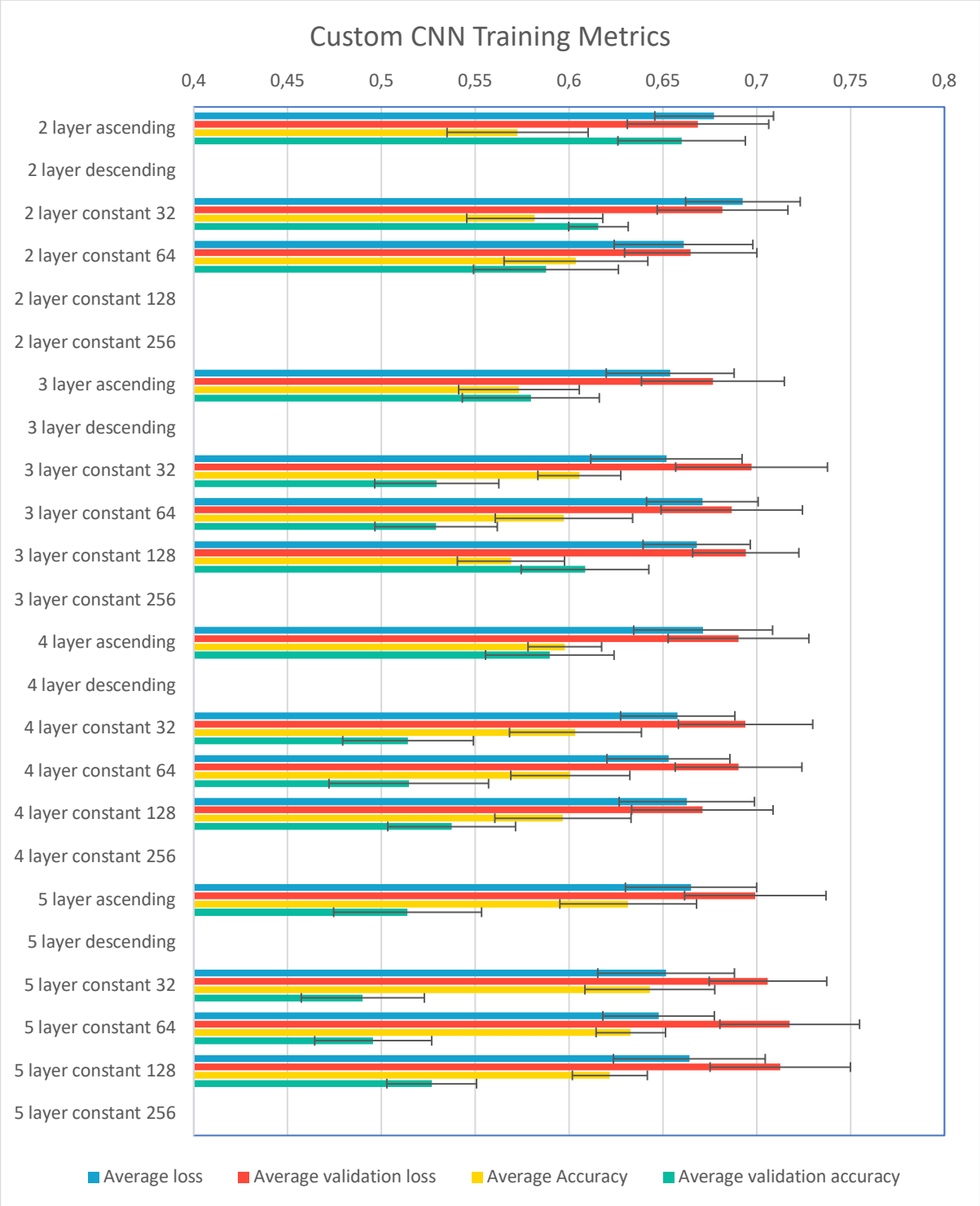

Figure 13: Average (colored rows) and standard deviation (error bars) of the metrics from the custom CNN training results. The individual metrics are denoted by the legend at the bottom. Note: Blank rows failed due to GPU memory limitations. Therefore, these will not be included in the comparison.

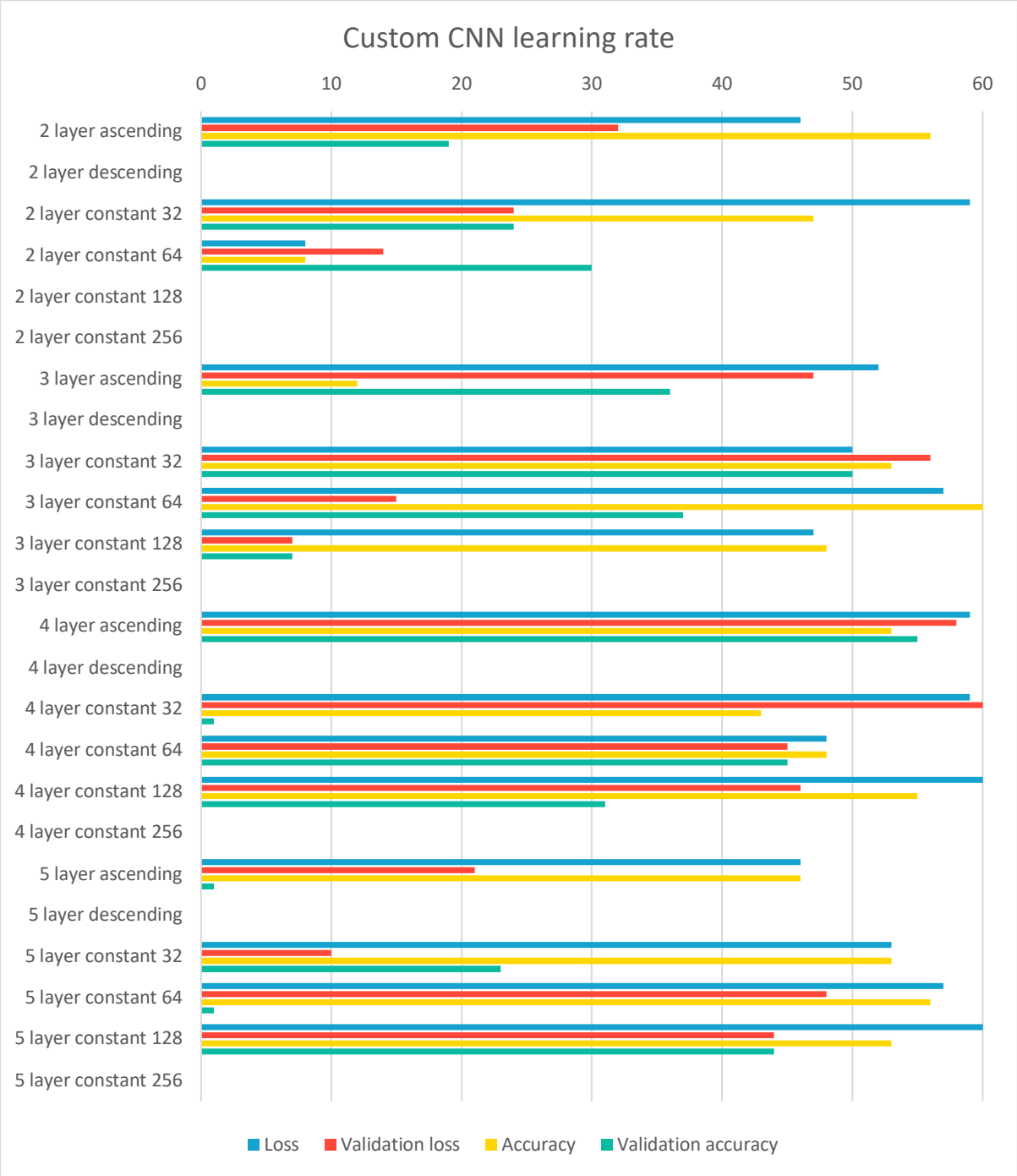

Figure 14: Convergence of the custom CNN models for individual metrics (shown in legend at the bottom), indicating at which epoch the optimal value was reached.

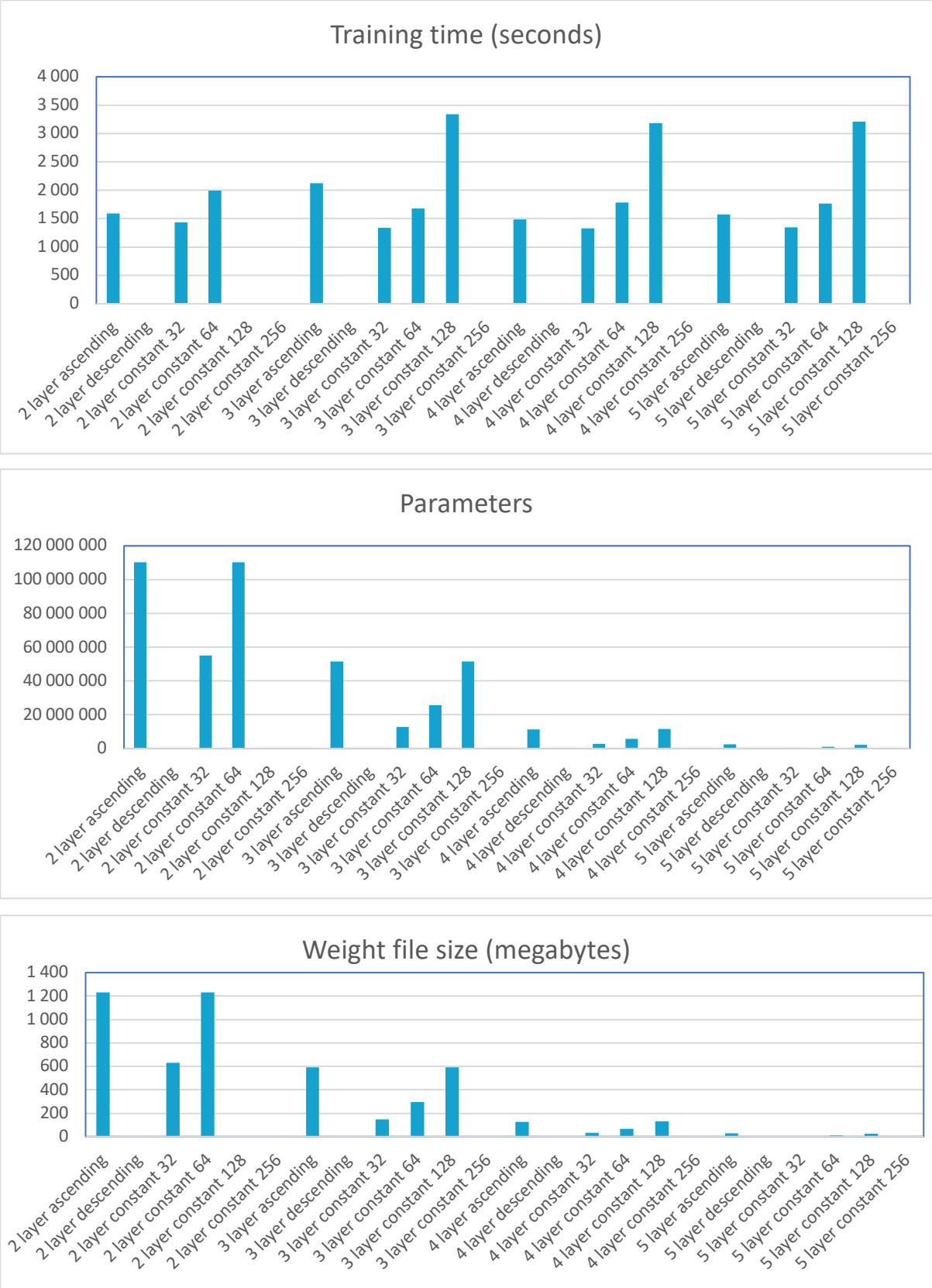

Figure 15: Additional results from the training of the custom CNN models. ***Top graph*** *compares the training time (for 60 epochs) in seconds, the* ***middle graph*** *compares the number of parameters and the* ***bottom graph*** *compares the size of the weight files in megabytes.*

2.2 Inception V3 CNN

Figure 16 and 17 shows the data from the transfer learning of the Inception V3 models.

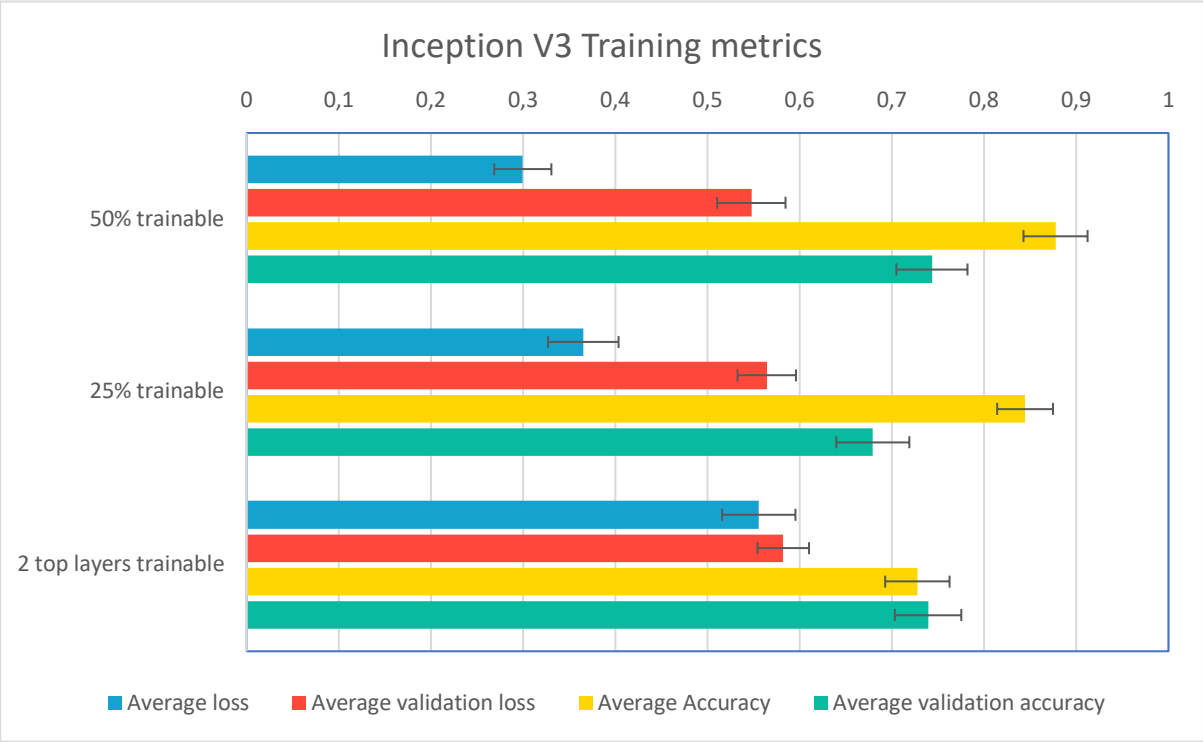

Figure 16: Average (colored rows) and standard deviation (error bars) of the metrics from the Inception V3 training results. The individual metrics are denoted by the legend at the bottom.

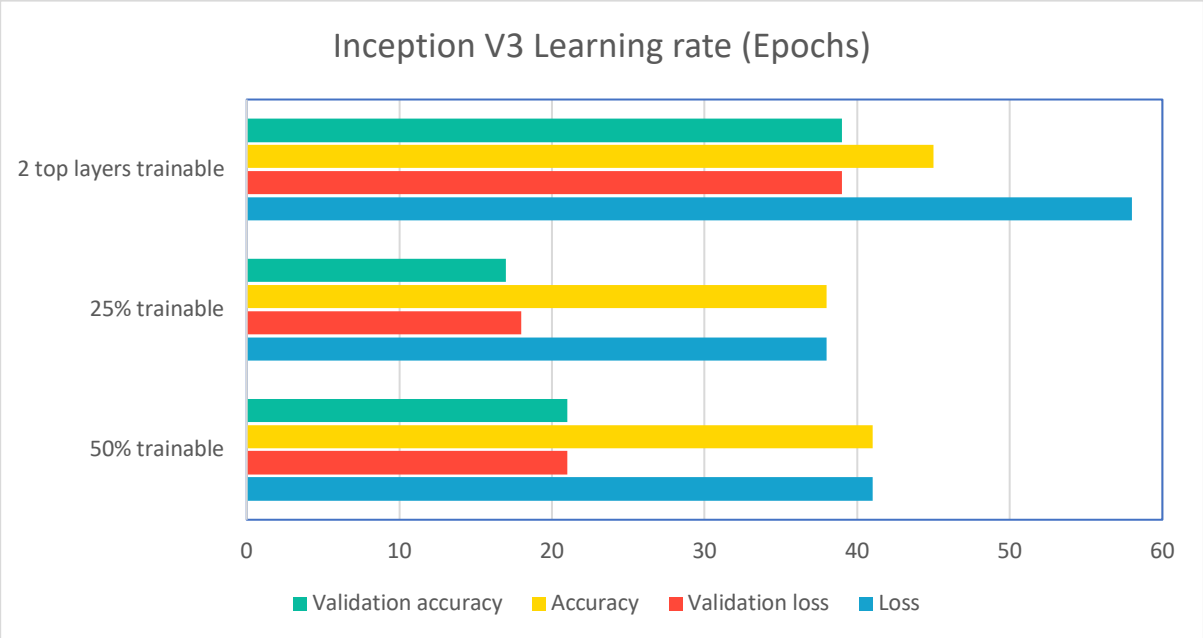

Figure 17: Convergence of the Inception V3 models for individual metrics (shown in legend at the bottom), indicating at which epoch the optimal value was reached.

All InceptionV3 training variants have **23 million parameters** and the weight file size is **163MB**.

2.3 LSTM

Figure 18, 19 and 20 shows the data from the training of the LSTM models.

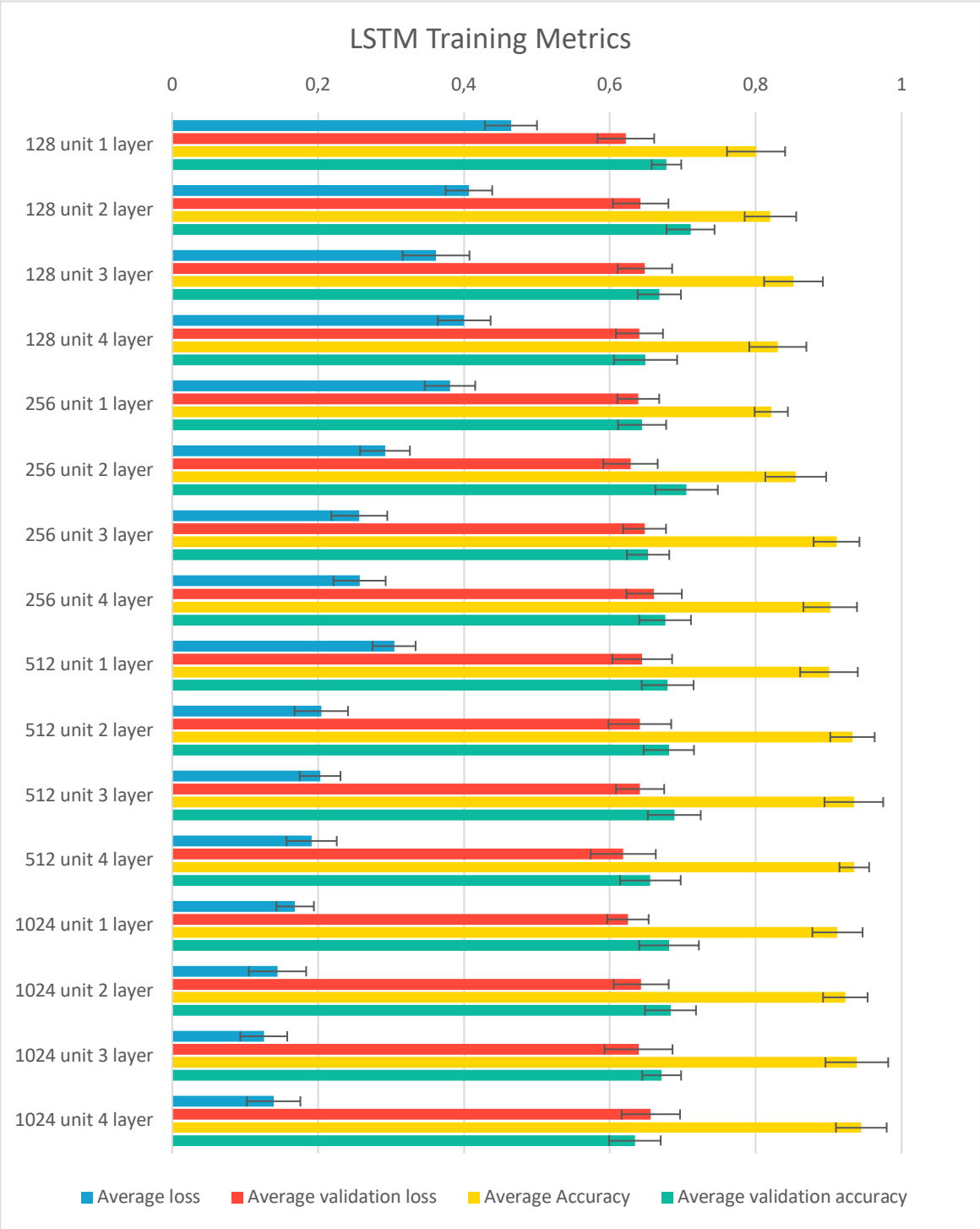

Figure 18: Average (colored rows) and standard deviation (error bars) of the metrics from the LSTM training results. The individual metrics are denoted by the legend at the bottom.

Figure 19: Convergence of the LSTM models for individual metrics (shown in legend at the bottom), indicating at which epoch the optimal value was reached.

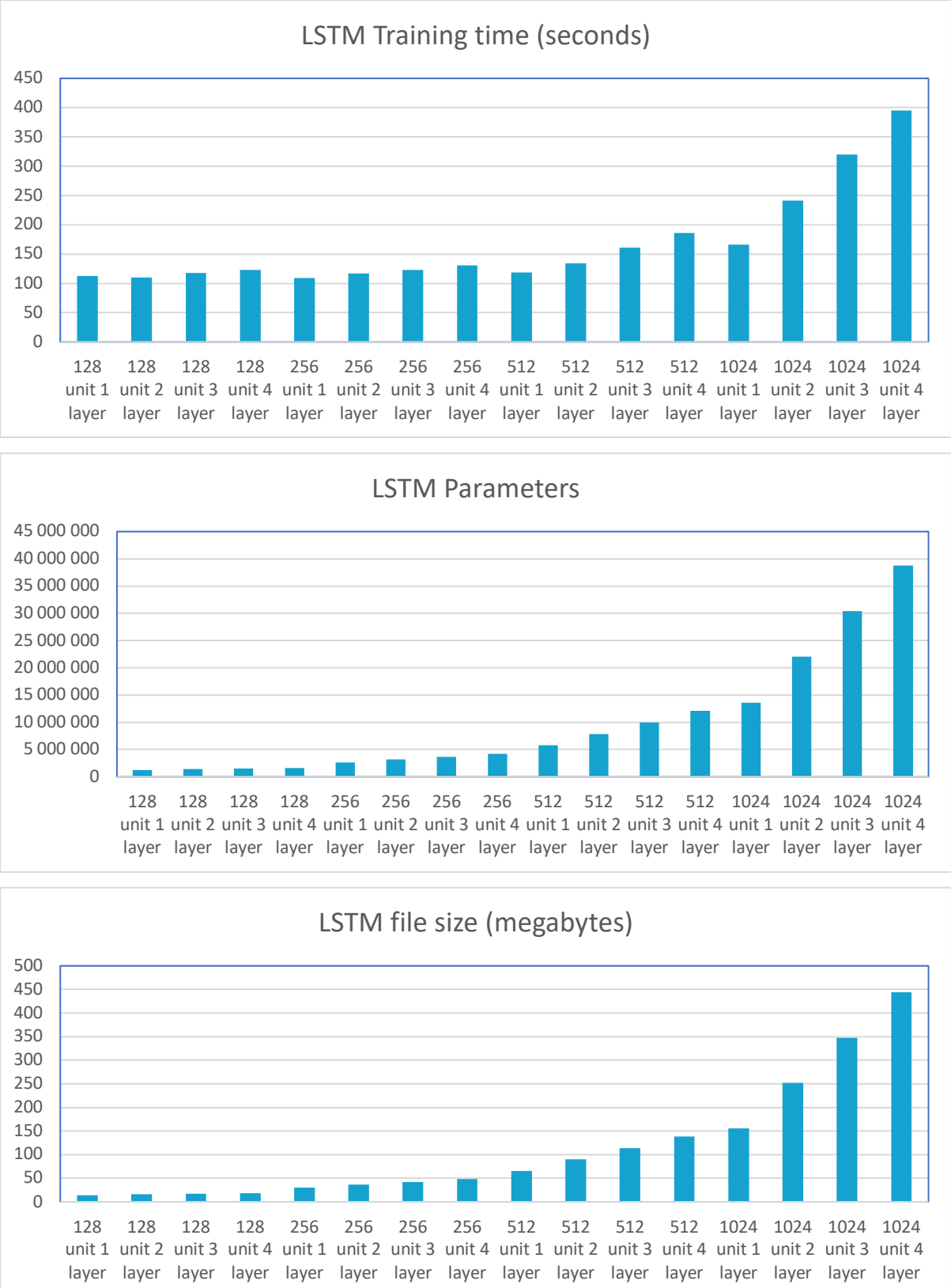

Figure 20: Additional results from the training of the LSTM models. ***Top graph*** *compares the training time (for 60 epochs) in seconds, the* ***middle graph*** *compares the number of parameters and the* ***bottom graph*** *compares the size of the weight files in megabytes.*

2.4 GRU

Figure 21, 22 and 23 shows the data from the training of the GRU models.

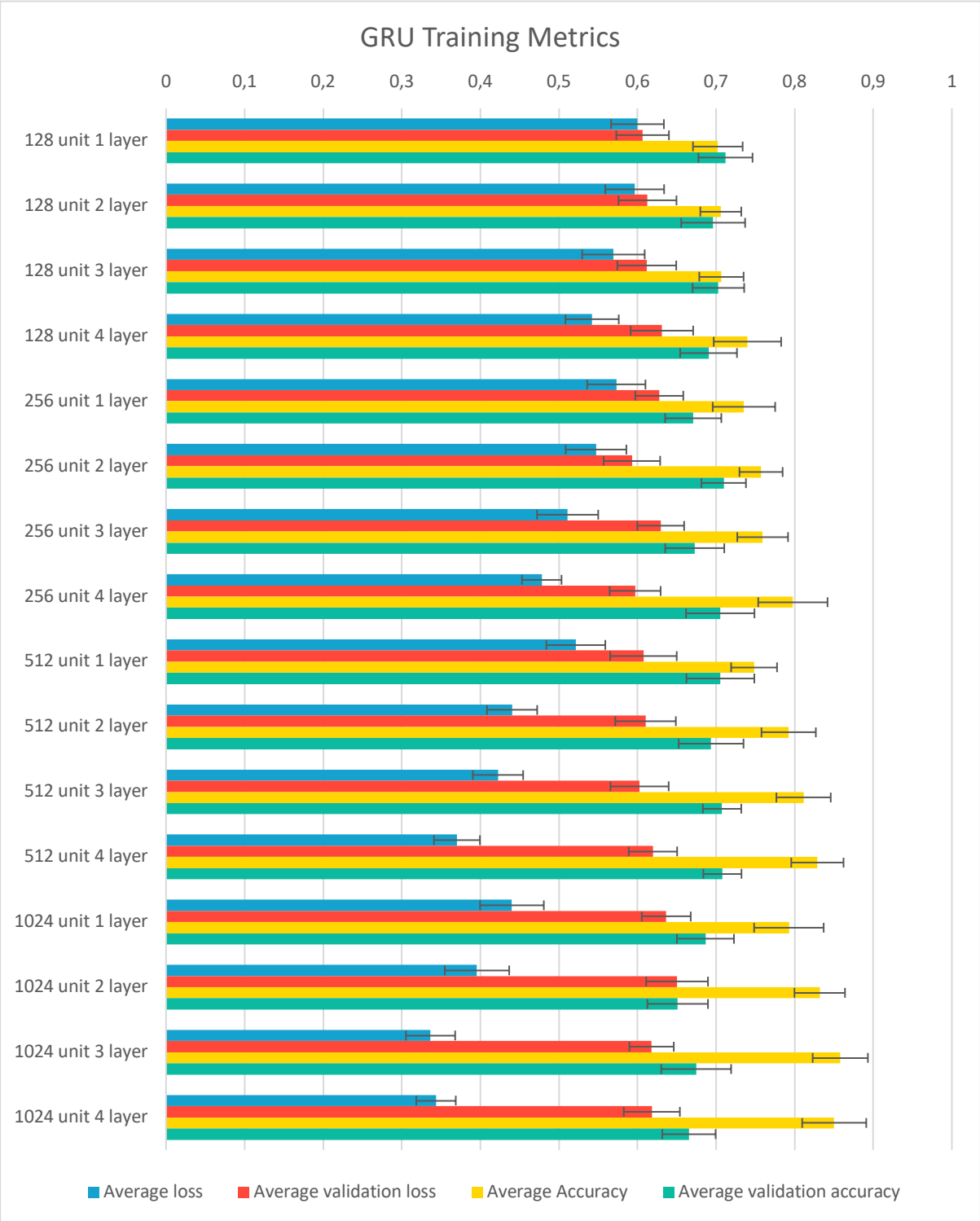

Figure 21: Average (colored rows) and standard deviation (error bars) of the metrics from the GRU training results. The individual metrics are denoted by the legend at the bottom.

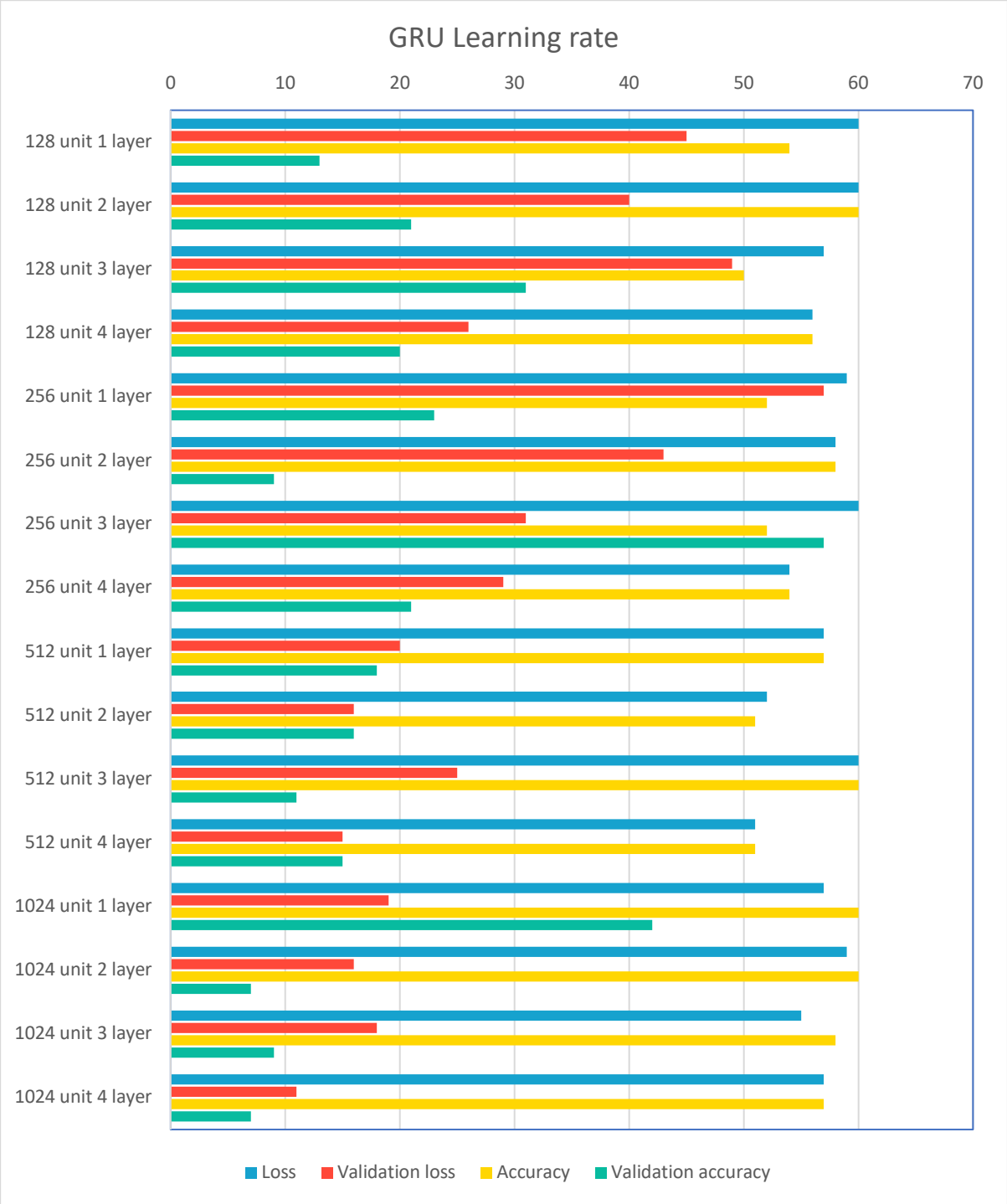

Figure 22: Convergence of the GRU models for individual metrics (shown in legend at the bottom), indicating at which epoch the optimal value was reached.

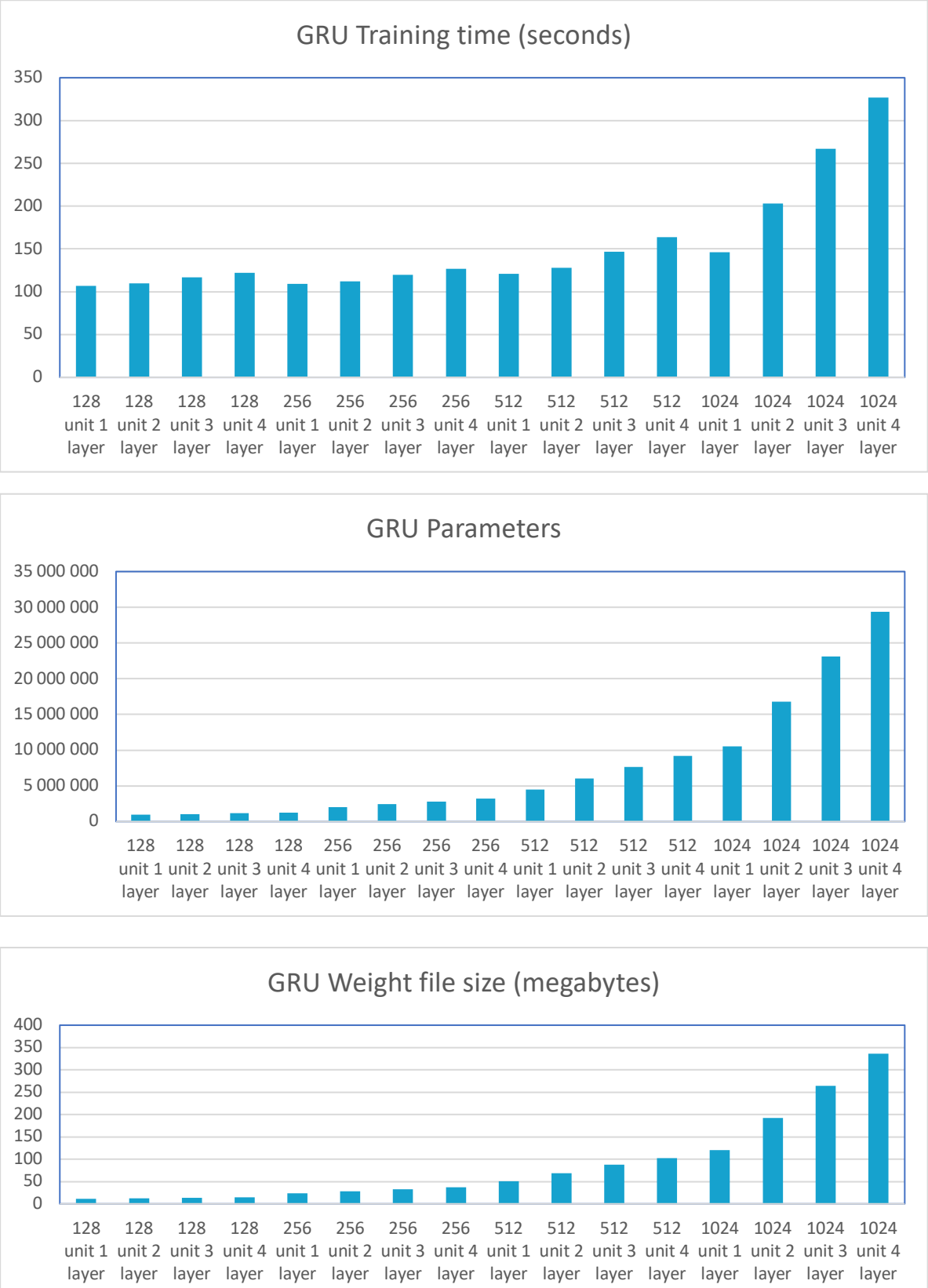

Figure 23: Additional results from the training of the GRU models. ***Top graph*** *compares the training time (for 60 epochs) in seconds, the* ***middle graph*** *compares the number of parameters and the* ***bottom graph*** *compares the size of the weight files in megabytes.*

2.5 MLP

Figure 24, 25 and 26 shows the data from the training of the MLP models.

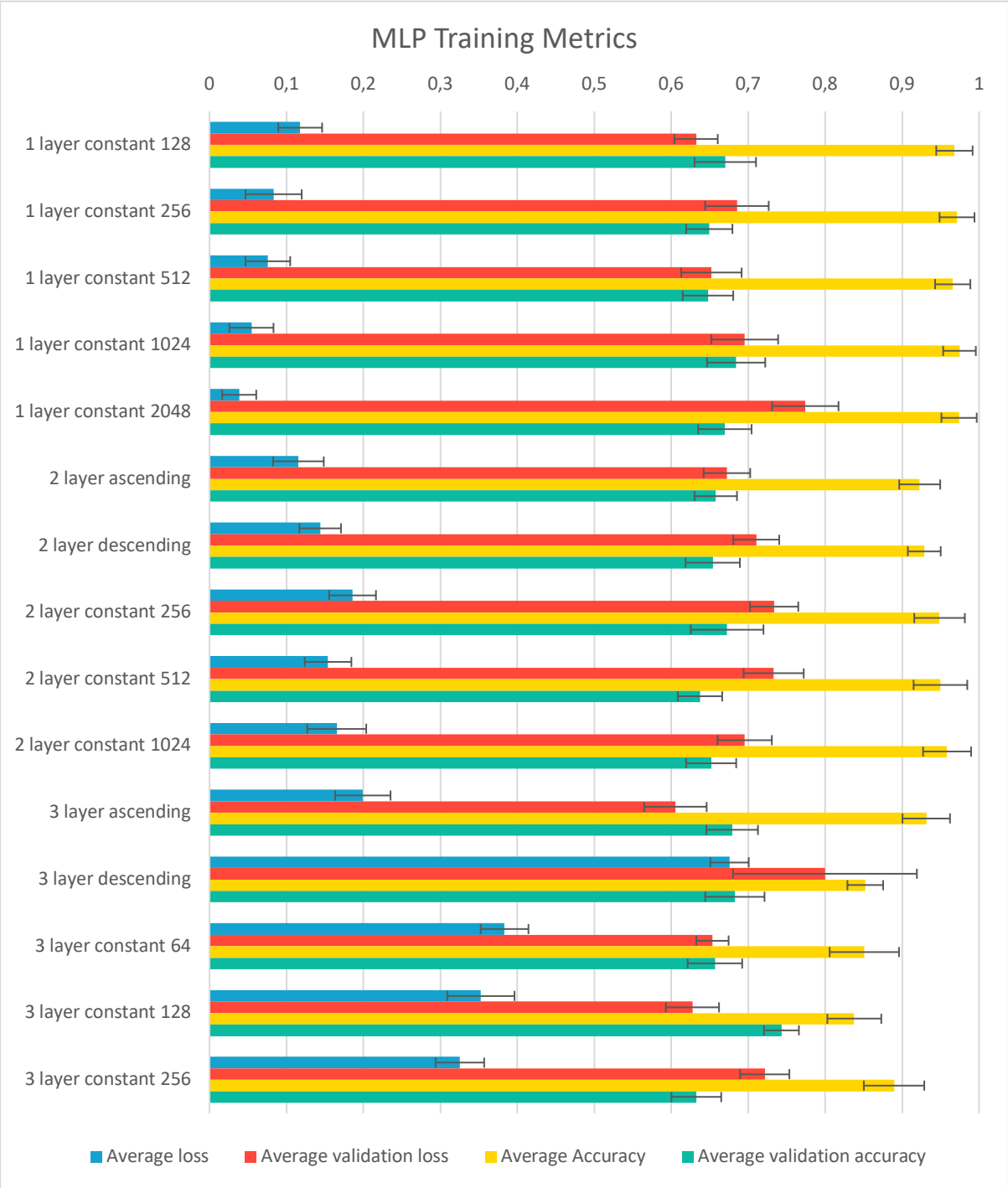

Figure 24: Average (colored rows) and standard deviation (error bars) of the metrics from the MLP training results. The individual metrics are denoted by the legend at the bottom.

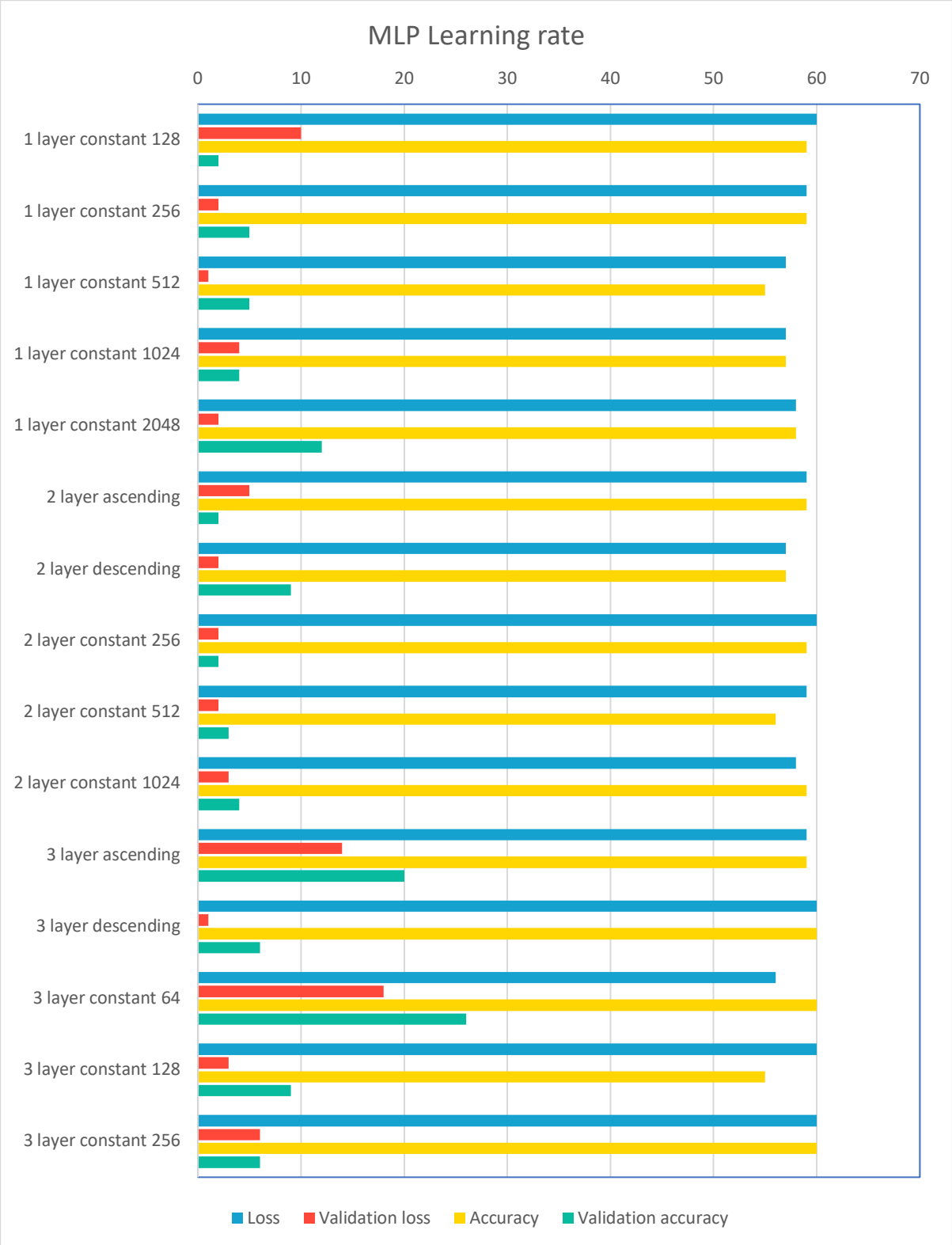

Figure 25: Convergence of the MLP models for individual metrics (shown in legend at the bottom), indicating at which epoch the optimal value was reached.

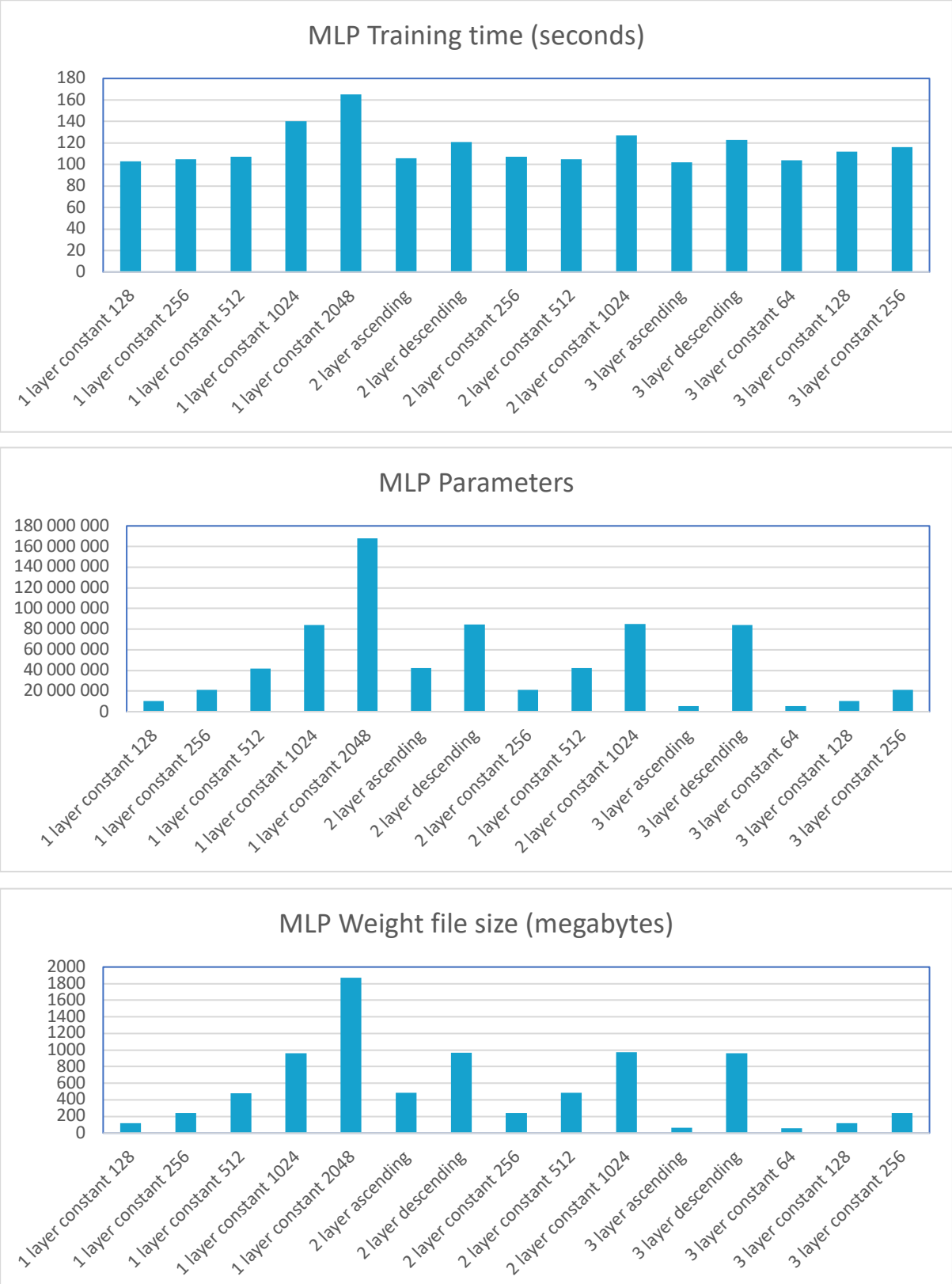

Figure 26: Additional results from the training of the MLP models. ***Top graph*** *compares the training time (for 60 epochs) in seconds, the* ***middle graph*** *compares the number of parameters and the* ***bottom graph*** *compares the size of the weight files in megabytes.*

3 – Discussion

3.1 Dataset limitations

While ideally all 2720 original/raw videos would be used to train the models, this was not feasible with the available hardware due to limited storage capacity. One approach to solve this was compressing the videos, significantly reducing the size and the quality of the videos. However, this reduction in details means losing crucial spatial information and resulted in poor performance metrics when testing models. Consequently, I chose to rather solve the issue by reducing the number of videos used to 1000.

A similar storage problem arose when initially extracting every individual frame from the full-length videos. Reducing the number of frames to 40 from each video is therefore necessary because it both conserves enough temporal information in the video, in addition to not being too large for the models to process. The 40 frames could either be all extracted from the start, end, middle or evenly spaced out in the video. I chose to extract evenly spaced out frames because it conserves more variation of the subject of the video.

The last limitation of the dataset was the excessive background information. Because the models attempt to only detect anomalies in the faces of the subjects, background noise can affect their performance. Cropping the frames to only include the faces therefore helps reduce this issue.

These data preprocessing steps mentioned above proved effective in early testing of the models (as compared to not preprocessing), having a significant impact on every performance metric.

3.2 Analysis

3.2.1 Custom CNNs

The depth of the models gives very different metrics. For shallower networks, multiple instances of the A_v were better than the A_t, indicating that the model generalized well for the unseen validation dataset. For instance, the **ascending units 2-layer model** has A_t=0.5725, while A_v=0.6599 (the highest of all custom CNNs). The same applies to the **constant 32 units**

2-layer model and the **constant 128 units 3-layer model** – all of which achieves the best three A_v (> 0.6). Deeper models significantly overfitted, with high A_t values and A_v values generally around 0.5 (indicating a random guess). In terms of the models' width, there were inconsistent results, and therefore it can be concluded that width does not have a significant impact on performance.

In regard to the L_r, all three models mentioned above learn relatively quick when compared to others. For example, the ascending units 2-layer model's L_v converges at epoch 32 and the A_v at 19. Additionally, the constant 32 units 2-layer model and the constant 128 units 3-layer model converges at epoch 24 and 7 for both validation metrics, respectively.

While the ascending units 2-layer model gives the best performance, the model's size is large with N_p=110 million and W_s=1.23GB. The constant 32 units 2-layer model mentioned has N_p=55 million and W_s=630MB, and the T_s=1430 seconds, which is one of the lowest compared to other models. Lastly, the constant 128 units 3-layer model is approximately equally sized, but T_s is approximately double the time.

Summarizing, while the ascending 2-layer model had better validation metrics compared to other mentioned models, its large parameter number and file sizes makes it unfeasible to realistically use. Therefore, because of its relatively good validation metrics and fast L_r, **the best overall custom CNN model is the constant 128 units 3-layer model**, despite the worse T_s.

3.2.2 Inception V3 CNN

The 50% trainable model achieves better results in all four metrics (L_t=0.2995, L_v=0,5475, A_t=0,8775, A_v=0,7434), as compared to the others. However, a measurement contradicting this is the L_r of the models, where the 25% trainable model approaches the best values of every metric approximately 3-4 epochs before the 50% trainable model. Nonetheless, this difference is insignificant when considering the relatively large performance difference in the models, and therefore it is reasonable to infer that **the 50% trainable model is most suitable**.

3.2.3 LSTM

Despite wider models having a greater difference between training (L_t and A_t) and validation (L_v and A_v) metrics, these models did not excessively overfit when compared to narrower models because every LSTM generalized relatively equally to unseen validation data. Two specifically well-performing models were the **128 units 2-layer model** (L_v=0.6423 and A_v=0.7106) and the **256 units 2-layer model** (L_v=0.6283 and A_v=0.7052).

When considering the L_r of the LSTM models, wider models converge faster on the validation metrics as compared to narrower models. In addition, for the wider models (e.g. 512 and 1024 units), the deeper variants converge at earlier epochs as compared to shallower models. Therefore, it is reasonable to conclude that deeper and wider models learn faster. When looking at the models mentioned above, the 128 units 2-layer model's L_v converges at epoch 15 and A_v at epoch 10. The 256 units 2-layer model converges 3-4 epochs slower for both metrics.

The 1024 units models generally perform worse in the T_s, N_p and W_s categories. The data therefore suggests that narrower models are significantly better in this regard. While the T_s of the 128 units 2-layer model and the 256 units 2-layer model are relatively equal, the former is approximately half the size with N_p=3.1 million and W_s=36.1MB.

Summarizing the discussion above, **the 128 units 2-layer model is the overall best performing LSTM model**. A counterargument for this could be the relatively slow learning rate, but such a tradeoff is necessary for the increased performance and versatility of the model.

3.2.4 GRU

The GRU models' L_v and A_v all performed relatively similar, indicating that every model generalized equally well for unseen data. However, increasing widths of the models correlates to significantly better training metrics (L_t and A_t). As with LSTMs, this does not necessarily mean that wider models were overfitting. The data does not reveal any noteworthy correlation between depth and performance. Despite this, two exceptions that suggest shallow and narrow models perform better are the **128 units 1-layer model** (L_v=0.6065 and A_v=0.7119) and the **256 units 2-layer model** (L_v=0.5928 and A_v=0.7096), which have the best A_v and L_v values, respectively.

When examining the L_r of the models, clear patterns emerge. For example, wider models generally converge on the optimal validation metrics quicker as compared to narrower models. The 128 units 1-layer model's L_v converges at epoch 45 and its A_v at epoch 13, while the 256 units 2-layer model's L_v converges at epoch 43 and its A_v at epoch 9. Therefore, based on both of these metrics the latter model is better.

As with the LSTMs, wider GRU models result in a higher N_p, larger W_s and generally longer T_s. Among the mentioned models above, the 128 units 1-layer model is significantly smaller (N_p=970 thousand and W_s=11.1MB) than the 256 units 2-layer model (N_p=2.4 million and W_s=27.8MB), while the T_s is relatively equal.

Based on the arguments presented above, **the 128 units 1-layer model is the best performing GRU variant,** not only because of its strong validation metrics, but also because of its decent learning rate and light size.

3.2.5 MLP

The MLP models displays unique results compared to the LSTMs and GRUs. A very clear pattern emerges in the metrics, with large differences in the L_t and L_v, in addition to the A_t and A_v. This could indicate the model overfitting. Two notable models are the **ascending 3-layer model** (L_v=0.6054 and A_v=0.6792) and the **constant 128 units 3-layer model** (L_v=0.6275 and A_v=0.7432). These achieve the best L_v and A_v among the MLP models, respectively.

Comparing the L_r of the two models mentioned above, the result is clear. The ascending units 3-layer model's L_v and A_v converges at epoch 14 and 20, respectively. In contrast, the constant 128 units 3-layer model converges at epoch 3 and 9 – a significant difference and improvement.

Lastly, it is important to note that the size of the ascending 3-layer model (N_p=5.4 million and W_s=62MB) is smaller than the constant 128 units 3-layer model (N_p=10.5 million and W_s=120MB). The T_s of the two models are insignificantly different.

Using the comparisons mentioned above, a reasonable conclusion is that **the constant 128 units 3-layer model is the best MLP model** because of its high A_v and fast L_r. Regardless of the greater N_p and W_s, the model outperforms other variants overall.

3.2.6 Summary

The best model from each DNN variant is included in the table 3 below.

Final comparison of results	Custom CNN Constant 128 units 3-layer	InceptionV3 50% trainable	LSTM 128 units 2-layer	GRU 128 units 1-layer	MLP constant 128 units 3-layer
L_t	0.6679	0.2995	0.2917	0.5999	0.3526
L_v	0.6941	0.5474	0.6283	0.6064	0.6275
A_t	0.5689	0.8774	0.8548	0.7021	0.8378
A_v	0.6085	0.7434	0.7052	0.7119	0.7432
L_r (for L_v)	7	21	18	45	3
L_r (for A_v)	7	21	14	13	9
Individual N_p	51.7 million	23.9 million	3.1 million	0.9 million	10.5 million
Total N_p	51.7 million	23.9 million	27 million	24.8 million	34.4 million
Individual W_s	591MB	163MB	36.1MB	11.1MB	120MB
Total W_s	591MB	163MB	199.1MB	174.1MB	283MB

Table 3: Summary and comparison of the best models of each architecture. The "Individual" and "Total" N_p and W_s are relevant for the models that use a feature extraction model in addition to their own architectures (LSTM, GRU and MLP). To get a realistic idea of size, the entire model – including the size of the InceptionV3 feature extractor – is listed under the "Total N_p" and "Total W_s" rows.

Comparing the best performing DNNs, it is easier to identify the strengths and weaknesses of individual architectures. Based on the training and validation metrics, it is clear that the InceptionV3 model performed the best in almost every category. While other models had relatively similar results in these categories, the size of the Inception V3 model is also an advantage. In terms of L_r, the custom CNN model was the best (with both L_v and A_v converging on epoch 7). Therefore, when training on limited amounts of data, the custom CNN model is most feasible to use.

Nonetheless, based on the discussion above, **the transfer learning technique with the Inception V3 CNN architecture is the best overall model for classifying real videos and deepfakes**, because it provides the optimal balance between every performance metric.

3.3 Conclusion

In conclusion, I have systematically investigated the optimal deep learning architecture for the task of detecting deepfakes. Experimentally testing variants of custom CNNs, pre-trained CNNs, LSTMs, GRUs and MLPs, the models were compared using specific performance metrics, and strengths and weaknesses of models were identified. Overall, the best model, successfully detecting 74% of the unseen data (A_v), was achieved by retraining half the layers in the InceptionV3 CNN architecture.

This discovery is significant because it proved which deepfake detection architecture has the most potential – CNNs – among the tested models. However, these findings have significant limitations. For example, the experiment only tested certain DNN architectures, and more extensive and broader testing might yield different results. Another limitation is the restricted variations of width and depth tested for every model. More variations within individual architectures could better illuminate clear patterns for what combinations are effective or not. Lastly, another limitation is the reduced amount of data, and therefore also the potential that the data does not represent real-life examples of deepfakes as accurately as possible. This could dramatically reduce the effectiveness of the models in practice.

Limiting the influence of deepfakes is vital, and every effort contributes. My hope is that this essay can provide a deeper insight into the difficulties involved in this detection, while also illuminating how to overcome this and successfully improve approaches to detecting deepfakes.

Bibliography

[1] Wiltz, Chris. “Deepfakes: The Looming Threat Of 2020.” *Designnews.com*, 6 Jan. 2020, www.designnews.com/artificial-intelligence/deepfakes-looming-threat-2020.

[2] Westerlund, Mika. “The Emergence of Deepfake Technology: A Review.” *Technology Innovation Management Review*, Nov. 2019, timreview.ca/article/1282.

[3] Adee, Sally. “What Are Deepfakes and How Are They Created?” *IEEE Spectrum*, 29 Apr. 2020, spectrum.ieee.org/tech-talk/computing/software/what-are-deepfakes-how-are-they-created.

[4] Harwell, Drew. “Top AI Researchers Race to Detect 'Deepfake' Videos: 'We Are Outgunned'.” *The Washington Post*, WP Company, 12 June 2019, www.washingtonpost.com/technology/2019/06/12/top-ai-researchers-race-detect-deepfake-videos-we-are-outgunned/.

[5] Chesney, Robert, and Danielle Keats Citron. “Deep Fakes: A Looming Challenge for Privacy, Democracy, and National Security.” *SSRN Electronic Journal*, 14 July 2018, doi:10.2139/ssrn.3213954.

[6] *Neural Networks Part 1: Setting up the Architecture*, lecture notes, CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, spring 2020, https://cs231n.github.io/neural-networks-1/

[7] Deeplizard, “Bias in an Artificial Neural Network explained | How bias impacts training”. Youtube, published 17. April 2018, retrieved August 25, 2020 from https://www.youtube.com/watch?v=HetFihsXSys

[8] McGonagle, John et al. Artificial Neural Network. Brilliant.org. Retrieved 21:02, August 25, 2020, from https://brilliant.org/wiki/artificial-neural-network/

[9] Kavlakoglu, Eda. “AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?” *IBM*, 27 May 2020, www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

[10] Lecun, Yann, et al. “Deep Learning.” *Nature*, vol. 521, no. 7553, 27 May 2015, pp. 436–444., doi:10.1038/nature14539.

[11] Li, Fei-Fei et. al *“Lecture 2 | Image Classification”*, CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, published 11. August 2017, retrieved 13. October 2020 from https://www.youtube.com/watch?v=OoUX-nOEjG0

[12] Shafkat, Irhum. “Intuitively Understanding Convolutions for Deep Learning.” *Medium*, Towards Data Science, 7 June 2018, towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1.

[13] Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[14] Wu, Jianxin. “Introduction to convolutional neural networks”, National Key Lab for Novel Software Technology. Nanjing University, China. Published May 1, 2017, https://cs.nju.edu.cn/wujx/paper/CNN.pdf

[15] Amidi, Afshine, and Shervine Amidi. “Convolutional Neural Networks Cheatsheet.” *CS 230 - Convolutional Neural Networks Cheatsheet*, Stanford University, 2019, stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks.

[16] *Convolutional Neural Networks: Architectures, Convolution / Pooling Layers*, lecture notes, CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, spring 2020, https://cs231n.github.io/convolutional-networks/

[17] Arora, Raman, et al. “Understanding Deep Neural Networks with Rectified Linear Units.” *ArXiv.org*, 28 Feb. 2018, arxiv.org/abs/1611.01491.

[18] Olah, Christopher. *Understanding LSTM Networks*, 27 Aug. 2015, colah.github.io/posts/2015-08-Understanding-LSTMs/.

[19] Sherstinsky, Alex. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network.” *ArXiv.org*, 31 May 2020, arxiv.org/abs/1808.03314.

[20] “Sequence Modeling: Recurrent and Recursive Nets.” *Deep Learning*, by Ian Goodfellow et al., MITP, 2018, pp. 367. (Placeholder1)

[21] Amidi, Afshine, and Shervine Amidi. “Recurrent Neural Networks Cheatsheet Star.” *CS 230 - Recurrent Neural Networks Cheatsheet*, 2019, stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[22] Hochreiter, S, and J Schmidhuber. “Long short-term memory.” *Neural computation* vol. 9,8 (1997): 1735-80. doi:10.1162/neco.1997.9.8.1735

[23] Cho, Kyunghyun, et al. “Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation.” *ArXiv.org*, 3 Sept. 2014, arxiv.org/abs/1406.1078.

[24] Phi, Michael. “Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation.” *Medium*, Towards Data Science, 28 June 2020, towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[25] Kostadinov, Simeon. “Understanding GRU Networks.” *Medium*, Towards Data Science, 10 Nov. 2019, towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

[26] Ebersole, Mark. “What Is CUDA?” *The Official NVIDIA Blog*, 10 Sept. 2012, blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/.

[27] Rössler, Andreas, et al. “FaceForensics++: Learning to Detect Manipulated Facial Images.” *ArXiv.org*, 26 Aug. 2019, arxiv.org/abs/1901.08971.

[28] Dufour, Nick, and Andrew Gully. “Contributing Data to Deepfake Detection Research.” *Google AI Blog*, 24 Sept. 2019, ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html.

[29] “Face Recognition”, Adam Geitgey, GitHub code repository, 2017, https://github.com/ageitgey/face_recognition

[30] Milton-Barker, Adam. “Inception V3 Deep Convolutional Architecture For Classifying Acute...” *Intel AI Developer Program*, 17 Feb. 2019, software.intel.com/content/www/us/en/develop/articles/inception-v3-deep-convolutional-architecture-for-classifying-acute-myeloidlymphoblastic.html.

[31] Szegedy, Christian, et al. “Rethinking the Inception Architecture for Computer Vision.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, doi:10.1109/cvpr.2016.308.

[32] Kurama, Vihar. “A Guide to ResNet, Inception v3, and SqueezeNet.” *Paperspace Blog*, Paperspace Blog, 5 June 2020, blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/.