Classification of human activity using neural network-based classifiers

Preparer: Alexey Pankratov

Introduction

The project aims to build single and multi-layer neural network-based classifiers of human activity and compare the performance of these classifiers. The dataset, on which these classifiers are trained and tested, contains the movement data gathered from 30 individuals using accelerometer and gyroscope embedded in the smartphones that those individuals were wearing on their waist while performing six activities - WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING. The data set features include the raw data of the raw 3-dimensional time-domain signals registered by gyroscope and accelerometer in 128 readings.

The dataset has already been split between Training sets represents 70% of the dataset, or 21 individuals, and the test set comprising the remaining 30%, or 9 individuals. Before building the classifiers, I split the training dataset further into training and validation sets in 80/20 proportion.

Building the NN-based classifiers

As mentioned above, I will build single- and multi-layer neural network-based classifiers in this project. In addition to different architecture, some less complex classifiers will be built on the dataset of body acceleration signals in 3 dimensions (3 features). Whereas other, more complex classifiers will be based on the dataset that also includes angular velocity vector measured by the gyroscope and total acceleration signal from the smartphone accelerometer in 3 dimensions (9 features).

Single-layer classifier

First, I will build a neural network with a single layer based on the dataset with three features. This classifier represents a transformation of each observation of the original dataset (input layer) by the SoftMax activation function. The SoftMax function takes as input a vector z of M real numbers and normalizes it into a probability distribution consisting of M probabilities proportional to the exponentials of the input numbers. That is, prior to applying SoftMax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying SoftMax, each component will be in the interval (0, 1), and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities. Accordingly, SoftMax function takes the observed body acceleration matrix as input and transforms each observation (i.e. row of the matrix with feature values) into a vector of 6 values, where each value corresponds to the probability that the observation is classified as one out of 6 labels that are mentioned in the introduction section. The label that has the largest probability is the predicted activity. The SoftMax function is defined as,

$$\sigma_c = \frac{e^{Z_c}}{\sum_{k=1}^{M} e^{Z_k}}$$

Where Z_c is a linear function of the input variables and I have a different linear function for each possible class our total M classes, or out of 6 activities in this project. For a single-layer neural network in this project, I can present Z_c as a linear combination $X\beta$, where X is the matrix of feature values (body acceleration), and β is the matrix of feature wights (including offsets). Building of this simple single-layer model is essentially an estimation of the abovementioned weights, which is done using the following cycle:

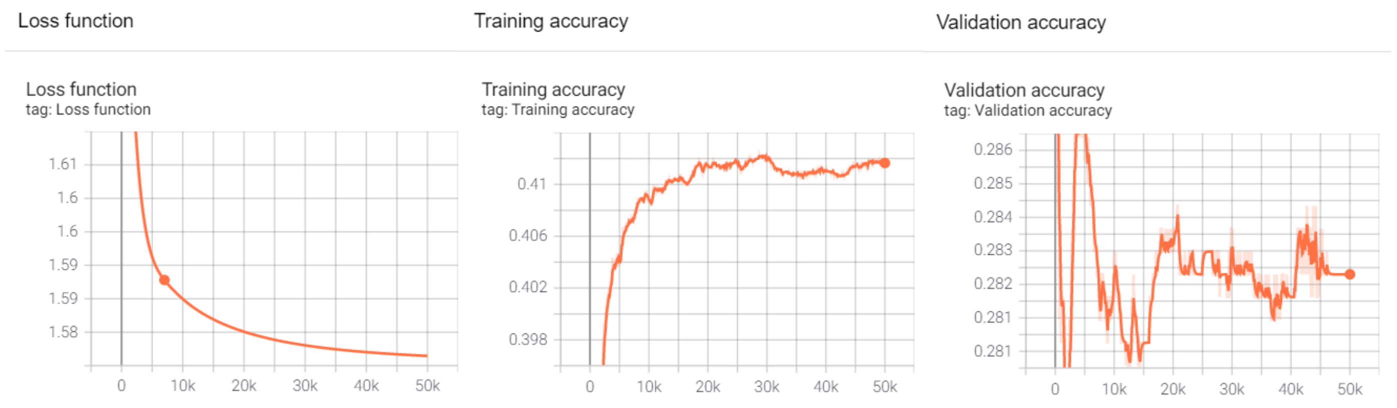
1. Take training feature set and calculate the probability of each activity with the initial/updated weights using SoftMax Function. For this project, I set initial weights equal to zero.
2. Calculate the deviance of predicted labels from observed training labels. This deviance is calculated using categorical cross-entropy loss function, which is defined as follows:

$$Loss = -\sum_{c=1}^{M} y_{o,c} log(\sigma_{o,c})$$

   Where M is the number of classes, $y_{o.c}$ is an indicator that is 1 if the class label c is the correct classification for observation o and 0 otherwise, and $\sigma_{o,c}$ is the result from SoftMax function from for class c for this observation.
3. The loss function calculated above tells us "how good" our model is at making predictions for a given set of features. So, the goal of a neural network is to optimize the loss function with respect to all parameters of the network. For the neural networks in this project, optimization means the minimization of the categorical cross-entropy loss functions with respect to the feature weights. This is done by an iterative optimization algorithm that is called gradient descent. There are different types of gradient descent based optimizers, and they will be discussed later in the report, but they all aim to find the global minimum of loss function by taking repeated steps in the opposite direction of the gradient of this function at the current point. The size of these steps is called the learning rate. As a result of the execution of the gradient descent, neural network weights are updated, and the cycle starts again from Step 1 but with updated weights.

I have run the cycle explained above 50,000 times (epochs) using a learning rate of 0.001 and Adam gradient descent optimizer (choice of the optimizer is explained later in this report). Performance metrics of the model are presented in Figure 1.

Figure 1 – Change in loss function, training, and validation accuracy for single layer function (extract from Tensorboard)



The single-layer neural network does not show high accuracy. Its test accuracy flattens at approximately 41.1%[1] after 20,000 epochs, and its validation accuracy is even lower – approximately 28.3%. Looking at the plot of change in the loss function, I can see that it also flattened and decreases very slowly, so I can say that the model is converged, and this is the best performance this model can achieve.

Low accuracy is most likely due to the single-layer neural network classifier is very simple. The original training set consists of 5,881 recordings of body acceleration in 3 dimensions made over 128 readings, so it has the shape of [5,881; 128; 3]. The low accuracy of the single-layer classifier may be due to this classifier does not fully use the structure of the training set. For the feature matrix to be used in this classifier, it has to be collapsed from 3 into 2 dimensions - [5,881; 384]. Multilayer neural network-based classifiers that are considered later in the project report fully use the structure of the data.

[1] The accuracy rates are presented in percentages in this report

Multi-layer classifiers trained on body acceleration data (3 features)

Now I will build more complex classifiers that have multiple layers. The way multi-layer neural networks are built is similar to the process described above –model parameters are estimated by optimizing the loss function using gradient descent and the same loss function is used. The difference in the multi-layer model compared to the single-layer one is in the number of layers. In this project, I will build sequential neural networks, which means that the training data serves as an input to Layer 1. The output of Layer 1 goes as input into Layer 2. The output of Layer 2 goes as input into Layer 3 and so on until the final Output Layer will produce class probabilities.

I first build the model with the architecture defined in the project requirements (is called "default architecture" afterward). I will then modify parameters of the model such as batch size, learning rate, and the number of epochs in order to improve the performance of the model. After that, I will try to adjust the model's architecture to increase the classifier's validation accuracy. Default architecture includes input layer and five other layers in the following sequence:

1. Input layer. This layer is training data. Unlike the single-layer model, the structure of the matrix of features is preserved in multi-layer neural networks, so its size is [5,881; 128; 3] as described above.
2. 1-D convolution layer with a kernel size of 4 and 32 filters. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the filter entries and the input at every spatial position.
3. Batch Normalization layer. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. This has the effect of stabilizing the learning process and reducing the number of training epochs required to train deep networks.
4. ReLu activation layer. The rectified linear activation function, or ReLU for short, is a piecewise linear function that will output the input directly if it is positive; otherwise, it will output zero.
5. GlobalAveragePooling1D layer. Global average pooling operation for the data. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps. Average Pooling calculates the average value for each patch on the feature map.
6. Dense layer with the SoftMax activation function. This layer is the same as I have already described in the single-layer neural network in the previous section of the report.

The critical consideration in building the neural network is which gradient descent-based optimizer to choose. As has already been mentioned above, there are different types of gradient descent-based optimizers, but they all aim to find the global minimum of loss function by using the gradient of this function. However, these optimizers differ in the ways that they respond to this gradient. I will first look at the most common types of gradient descent-based optimizers and then chose the most appropriate for this task:

- Batch gradient descent. This algorithm computes the instantaneous gradient of the cost function with respect to the model parameters for the entire training dataset. Considering that this algorithm uses all training data at once, it is very computationally expensive; therefore, its use is not practical. Also, as this algorithm uses the entire training dataset, it does not introduce noise in the model parameter estimates, making it more likely that I will get stuck in the local minimum of the loss function rather than its global minimum.
- Stochastic gradient descent (SGD). Stochastic gradient descent aims to overcome the drawbacks of batch gradient descent. Instead of performing computations on the whole dataset — which is redundant and inefficient — SGD only computes on a small subset of random selection of data examples, thus introducing the noise in the model parameter estimates.
- Adaptive Moment Estimation (Adam). It includes a changing learning rate and a momentum term. The optimizer keeps a running average of the recent gradients and squared gradients; effectively, it remembers the values of the gradients experienced, with the hyperparameters controlling the length of its memory. Compared to two Gradient Descent optimizers described above, Adam computes not instantaneous but average gradient of the recent history. This allows Adam to introduce the momentum into gradient descent and allow parameters to change rapidly and cross local minima without getting trapped. Also, the learning

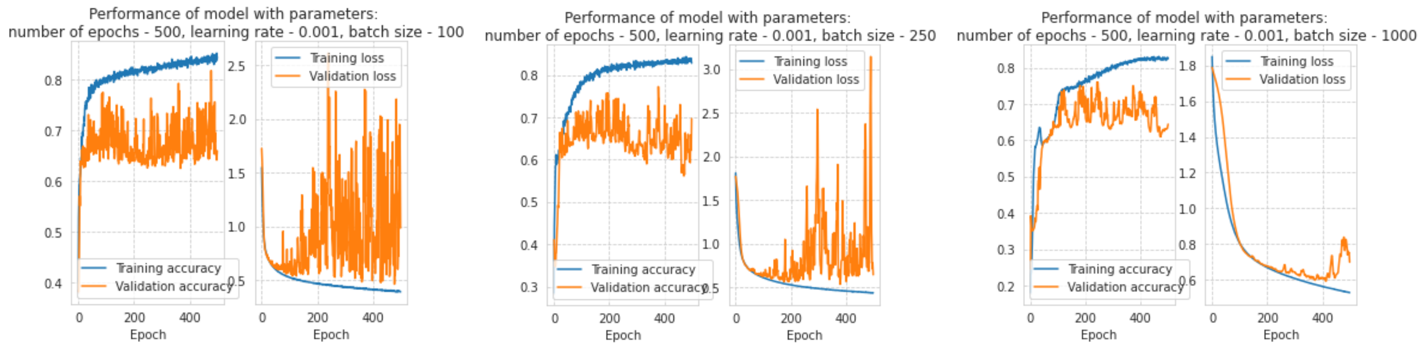
rate in the Adam optimizer is divided by the square root of the average squared gradient, which ensures that each step is approximately the same size. If the second moment is large, this term will reduce the step size.

I can see that Adam optimizer adopts better strategies to avoid the local minima. Also, Adam adapts the learning rate to the rate of the descent. There are other optimizers such as RMSprop and Adadelta, but the way they work is similar to Adam optimizer. Adam is one of the most widely used and practical optimizers for training deep learning models so consider it to be the best overall choice.

Next, I will look at the impact of the choice of batch size on the model performance. Batch size is the number of training examples utilized in one iteration. In contrast, one epoch is when an entire training dataset is passed through a learning cycle once. Therefore, the smaller the batch size, the more itrerations it takes for the training dataset to go through one learning cycle entirely. For example, the training dataset of this project contains 5,881 observations (see above), so it would take 59 iterations for the model with a batch size of 100 to go through 1 epoch (58 complete batches with 100 training examples + 1 batch with 81 training examples in it). I built three models with the same parameters except for the batch size. The performance results of each model are presented in Figure 2. As I can see, with the same learning rate, it takes more epochs for the model with larger batches to reach higher training accuracy than for the model with a smaller batch size. The model with a batch size of 1000 reaches 80% training accuracy approximately after 400 epochs. Whereas models with the batch size of 100 and 250 reach it approximately after 150-200 epochs. Also, I can see that the training loss of the models with 100 and 250 almost flattened, but the training loos of the model with 1,000 batch size is still decreasing.

On the other hand, as smaller batch size introduces more noise in the parameter estimates, valuation loss and valuation accuracy in the models with smaller batch sizes are much more variable than in the model with a batch size of 1000. It should be noted that neither of the models in Figure 2 shows good performance as I can see that validation loss increases and goes further away from the training loss. This is due to the initially chosen learning rate of 0.001 is too high. Decreasing the learning rate will solve the non-convergence problem, as I will see later. For this project I chose batch size of 250 as a compromise value as it both allows to flatten the loss function and accuracy more quickly then in case of the model with the larger batch size and at the same time does not make validation loss function too noisy as I will see below in the report.

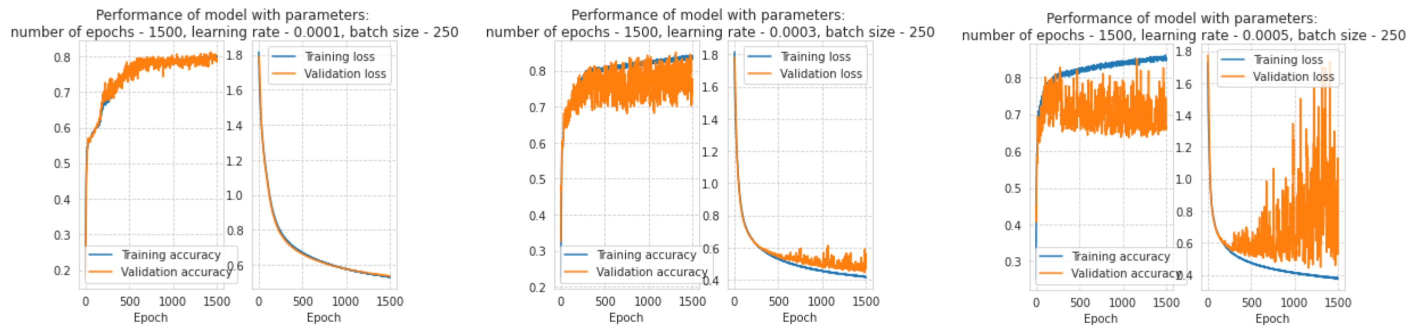
Figure 2 – Comparison of performance of multilayer neural networks with initial multilayer architecture of different batch sizes (batch size of 100, 250 and 1,000)



After the batch size is settled, I assess the optimal learning rate and the number of epochs. In order to do this, I build the models with different learning rates trained over the 1,500 epochs. I decided to take a large number of epochs to see at which point these models reach convergence. In other words, the model achieves a state during training in which loss settles to within an error range, i.e., the point where the loss flattens and changes only slightly after. I will also look at signs of model overfitting, if any, i.e., when the training performance metrics continue to improve (accuracy is growing and loss is decreasing) but validation metrics are starting to deteriorate. Overfitting means that a model has learned 'too much' from the training data, so it becomes extremely good at predicting it but fails to generalize these predictions for new data (validation and test).

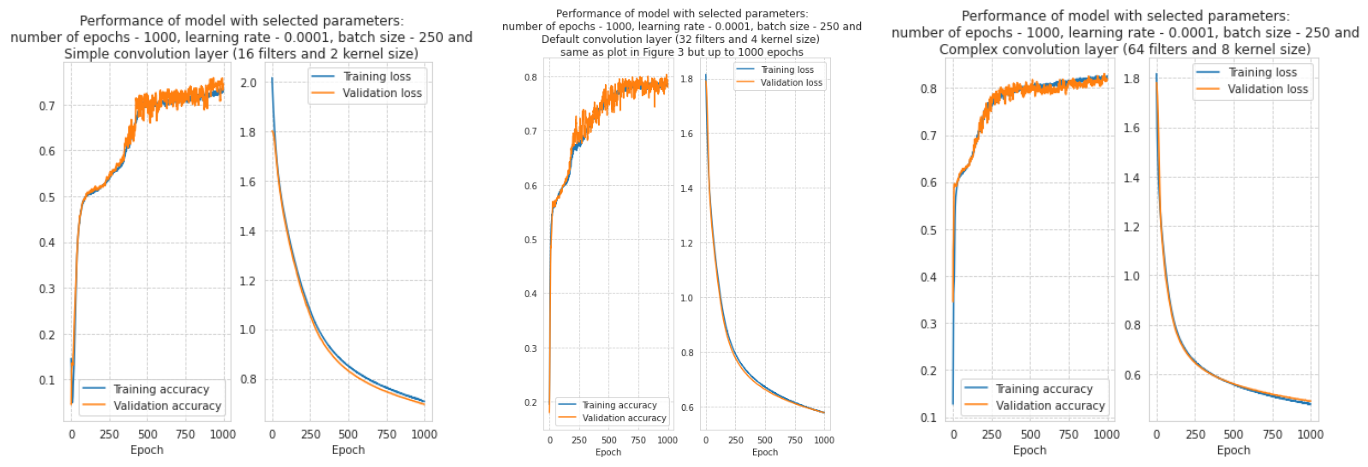
I have already seen in Figure 2 that the models did not reach convergence due too learning rate was too high. So I selected the models with the lower learning rates – 0.0005, 0.0003, and 0.0001. Figure 3 below shows the comparison of the neural networks with different learning rates. As I can see, validation loss does not go in line with the training loss in the models with learning rates of 0.0003 and 0.0005. It means that these rates are still too high. Whereas, the learning rate of 0.0001 is optimal – the validation loss plot is almost identical to the training loss, and validation accuracy, although a bit noisy, goes along with the training accuracy on the plot.

Figure 3 – Comparison of performance of multilayer neural networks with initial multilayer architecture run with different learning rates (learning rates of 0.0001, 0.0003 and 0.0005)



Considering the number of epochs, I can see in Figure 3 that the model with the learning rate flattens approximately after 1,000 epochs. After 1,000 epochs, improvements in training and validation accuracy and loss are very small to consider 1,000 epochs optimal.

Figure 4 – Comparison of performance of multilayer neural networks with different convolutional layers



Now, I will explore how different model parameters and architectures affect the validation accuracy. I will start by altering the kernel size and number of filters in the existing architecture. Plots that are presented in Figure 4 use selected parameters (batch size -250, learning rate - 0.0001, and the number of epochs -1,000) and amend the convolutional layer in the default architecture defined at the section's beginning. The right plot represents the model with the convolutional layer of lower complexity than the default one. The layer has 16 filters and a kernel size of 2. The middle plot represents the default case. It is the same as the left plot in Figure 3 but stopped at 1,000 epochs as this number of epochs was selected. And the right plot represents the model with a more complex convolutional layer with 64 filters and a kernel size of 8. Figure 4 shows that the convergence rate increases with the complexity of the convolutional layer. Whereas the model with the default convolutional layer (plot in the middle) converges only after approximately 1,000 epochs, the model with more complex convolutional layer does that earlier – between

500 and 750 epochs.[2] The model with a simple convolutional layer, on the other hand, has not yet converged, its loss function is still decreasing, and there is a steep slope of the accuracy curves at even the 1,000th epoch.

The model with the complex convolutional layer achieves 81.78% validation accuracy after 1,000 epochs. Now I will adjust the model architecture further to see if the validation accuracy can be further improved. I modified the model's architecture as follows:

Input layer >>> Transposed convolutional layer with 64 filters and kernel size of 8 >>>

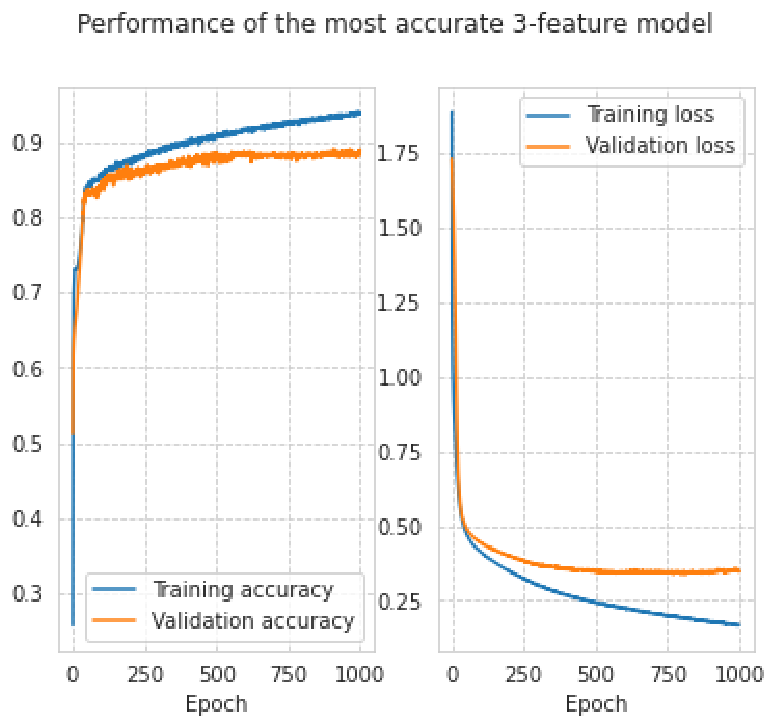
Batch normalization layer >>> Average pooling layer >>>

Convolutional layer with 64 filters and kernel size of 8 >>> ReLu activation layer >>>

Global max pooling layer >>> Dense layer with the SoftMax activation function

The above architecture was derived by try and error method to achieve the highest validation accuracy possible. Figure 5 shows the performance of the model with this architecture.

Figure 5 – Performance metrics of three features based neural network with architecture that gives the best validation accuracy.



Although the validation loss is above the training loss, which is a sign of overfitting, validation loss and accuracy flatten at approximately 0.35 and 89% respectively after 300 epochs and neither improve nor getting worse further the training process.

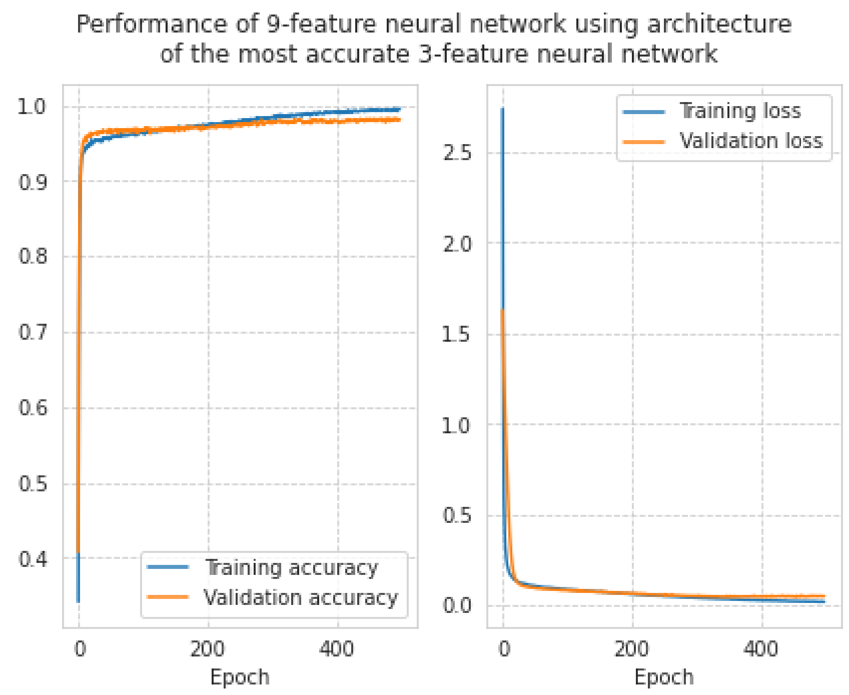
Validation accuracy of 89.06% is much above 81.78% accuracy achieved by the models with default architecture and more complex convolutional layer demonstrated above.

[2] Although the loss function continues to decrease in models with the default and more complex convolutional layers, the rate of the decrease is very small, so the model are considered to be converged.

Multi-layer classifiers trained on body acceleration data, angular velocity, and total acceleration data (9 features)

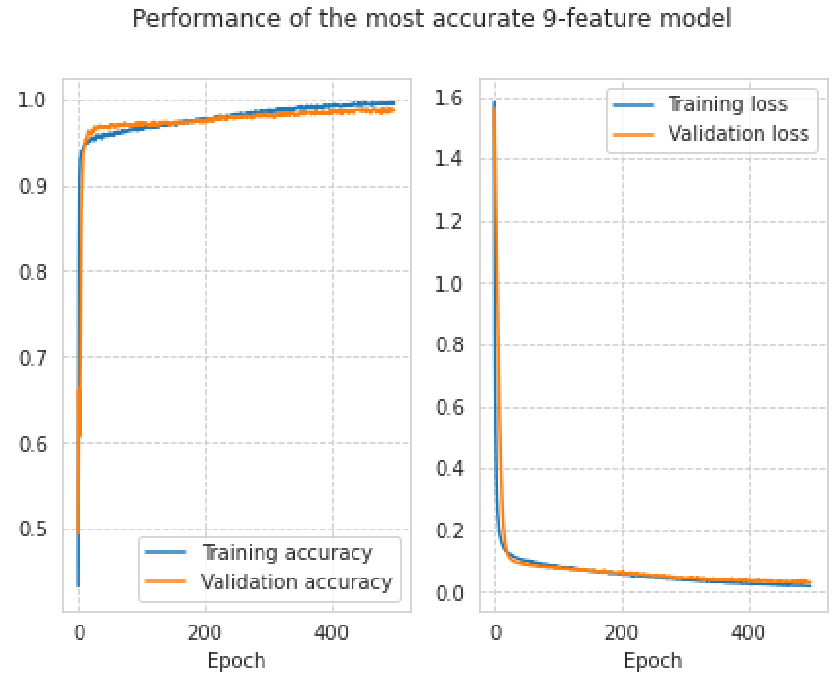
Finally, I will train the neural network on all signal data available. Same as three feature data, I first split the training set further into training and validation sets in 80/20 proportion, and then I built two models.

Figure 6 – Performance metrics of nine features based neural network with the same architecture that showed the best validation accuracy of three features based model in Figure 5.



The first network is built with the same architecture as the previous model (see Figure 5), with only the input layer modified to fit 9 feature training data. As it reached convergence very quickly, I changed the number of epochs to 500 to avoid overfitting. The performance metrics are presented in Figure 6. This model's training and validation performance vastly exceed the performance of the 3-feature neural network with the same architecture from Figure 5. In addition to higher validation accuracy (98.16% vs. 89.06% for the 3-feature model), the 9-feature model has a much smaller difference between training and validation loss, indicating better generalization.

Figure 7 – Performance metrics of nine features based neural network with architecture that gives the best validation accuracy.



And the final model is the model with all data and the adjusted architecture that provides the highest accuracy possible. The abovementioned model already showed very high validation accuracy. Still, it was possible to improve this metric further by changing the transpose convolutional layer in the previous architecture to the simple convolutional layer. Surprisingly, when I sought the best architecture by try and error method for the model with 3-features in the previous section, the model with such architecture performed worse than the one with the transposed convolutional layer, so transposed convolutional layer was preferred then. This is not the case for the neural network with 9-feature. Loss and accuracy improvement over the training cycle is presented in Figure 7. This model demonstrates a slightly better validation accuracy of 98.71% compared to 98.16% of the model above. As the validation accuracy of both models reached a plateau for the last 100 epochs of the training cycle, I can also compare average validation accuracy over this period to see if the above difference is purely due to noise or not. The average validation accuracy of the models in Figure 7 is 98.57%, which is also better than the validation accuracy of 98.05% of the model in Figure 6.

Comparison of performance on test data and conclusion

Finally, I compared the test accuracy of the three following models:

- Single-layer neural network-based classifier (Figure 1)
- Multilayer neural network-based classifier trained on body acceleration data (3 features) that gave the best validation accuracy (Figure 5)
- Multilayer neural network-based classifier trained on angular velocity, body, and total acceleration data (9 features) that gave the best validation accuracy (Figure 7).

The training, validation, and test accuracy of each model is presented in Table 1.

Table 1 – training, validation and test accuracy metrics of final neural network based classifiers.

	Training accuracy	Validation accuracy	Test accuracy
Single-layer NN	41.17%	28.28%	31.18%
Mutli-layer NN with 3 features input layer	93.78%	89.06%	83.47%
Mutli-layer NN with 9 features input layer	99.42%	98.71%	93.99%

The table above shows that the single-layer NN-based classifier performs much worse than multi-layer classifiers. Also, among the multi-layer classifiers, the model trained on all signals (9-feature input layer) outperforms the one trained only on body acceleration data (3-feature input layer). So, increasing the model complexity and the number of features allowed neural networks to improve the classification accuracy of human activities. At the same time, it should be noted that there is a drop in test accuracy compared to validation and training accuracy, which is not the case for the single-layer neural network. However, still, the difference in performance between single- and multi-layer networks is vast.

The test accuracy of the single-layer model that used extracted features from Assignment 1 was 95.59%. Although the model's architecture was of the same low complexity as the single-layer model in this project, it shows the largest test accuracy compared to all the models discussed in this report. The final model from Assignment 1 was based on 155 features, and it is likely that the model of higher complexity that takes these features as input would have achieved even better results. This example emphasizes that building the model is not only about defining the architecture of the neural network. It is crucial to understand the data, as performing proper feature engineering can further improve the classification accuracy.