Embedded Systems
DL model optimization for Lightweight Keyword Spotting

Team Members :-
Mistry Parth Kirsankumar (B20EE090)
Nagda Yash Narendra (B20EE092)

Abstract :

This project focuses on the optimization of a deep learning (DL) model for lightweight keyword spotting, which is the process of detecting specific keywords or phrases within an audio signal. The proposed approach aims to reduce the computational complexity of the DL model while maintaining high accuracy levels. This is achieved through various techniques such as model compression, pruning, and quantization. The optimized model is evaluated using a dataset of speech samples, and its performance is compared which varying the structure of the model.This research has implications for the development of efficient and lightweight keyword spotting systems that can be deployed on resource-constrained devices, such as smartphones and wearable devices.

Introduction :

Keyword extraction from audio signals is essential for a range of applications, including speech recognition, keyword spotting, and information retrieval. It involves the identification and extraction of specific words or phrases from an audio signal, which can then be used for various purposes, such as transcription, summarization, or categorization.

One of the primary needs for keyword extraction from audio signals is speech recognition, which involves converting spoken words into text. This is a critical technology that enables people to communicate with computers and smart devices through voice commands. Keyword extraction is a critical step in speech recognition as it involves identifying specific words or phrases that are of interest to the user, such as a question or a command.

Keyword spotting is another application of keyword extraction from audio signals, where the goal is to detect specific keywords or phrases within an audio signal.

About The Dataset :

Keyword Extraction using the audio signal dataset

- The Audio Signal dataset consists of the audio signal of various keywords.
- The dataset contains 35 different keywords that were supposed to be extracted.

Data Preprocessing :

Mel-Frequency Cepstral Coefficients (MFCC) is an extensively employed feature extraction method in speech recognition and audio processing.
MFCC preprocessing in deep learning entails transforming auditory signals into a set of feature vectors that can be used as input to a neural network. This procedure involves a number of steps:

- Pre-emphasis: to amplify the signal's high-frequency components.
- Framing: to divide an audio signal into tiny frames that overlap.
- Windowing: It is the process of applying a window function to each frame in order to reduce spectral leakage.
- Fourier transform: To transform each frame from the time domain to the frequency domain.
- Mel filterbank: to apply a filterbank that simulates the non-linear human auditory perception.
- Logarithmic compression: to reduce the dynamic range of the output of a filterbank.
- Discrete Cosine Transform: To derive the cepstral coefficients.

The resulting MFCC vectors contain information about the spectral content of the audio signal and are frequently employed as input features for tasks such as speech recognition, speaker identification, and music genre classification.

Deep Learning Model :

The imported dataset was then converted into tensors of batch size = 64. Later these were used to train and validate the deep learning model.
We particularly used feed forward neural network for this project. A feedforward neural network is a type of artificial neural network in which information flows from input to output through interconnected layers of neurons. The network employs weights to adjust the strength of the connections between neurons, and the weights are adjusted during training to minimise the difference between the predicted and actual output. Common applications of feedforward neural networks in machine learning include image classification, natural language processing, and speech recognition.

Parameters Used To Build Best Model :

Dense Layer : It is a type of layer in which every neuron in the layer is connected to every neuron in the layer beneath it. This indicates that each input is connected to each neuron in the dense layer, and each neuron in the dense layer is connected to each neuron in the following layer.

Activations Function : Activation functions are mathematical functions applied to the output of each neuron in a layer to introduce nonlinearity and enable the network to learn complex data patterns and relationships.
Following are the activation functions that we used.

- **Sigmoid Function** is a well-known activation function that converts any real-valued number to a value between 0 and 1. It is frequently employed for binary classification assignments.

- **ReLU (Rectified Linear Unit)** function is a straightforward yet effective activation function that returns the input if it is positive and 0 otherwise. It is computationally efficient and widely utilised in numerous DNN varieties.

- **Tanh (Hyperbolic Tangent)** function is comparable to the sigmoid function, but it maps input values to a range of -1 to 1. It is also employed in binary classification assignments and, in some instances, provides superior performance to the sigmoid function.

- **The softmax function** is frequently used in the output layer of a DNN for multi-class classification assignments. It translates the output of each neuron to a probability distribution over the classes, making sure that the sum of the probabilities equals 1.

Learning rate : It is a hyperparameter that regulates the size of the model weight update steps during training. It is a scalar value multiplied by the loss function gradient with respect to the weights to determine the direction and amplitude of the weight update.

Batch size : It is a hyperparameter that determines the number of samples analysed during each training iteration. During training, the input data is divided into batches, and each batch is transmitted to the model for forward and backward propagation in order to update model parameters.

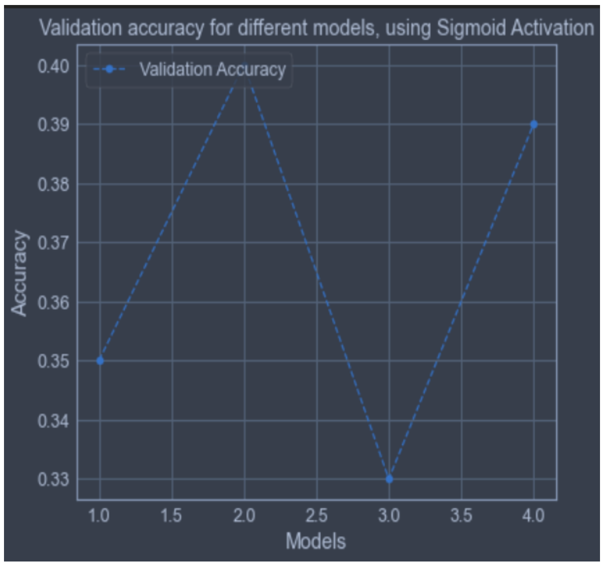
Different Deep Learning Models & Analysis :

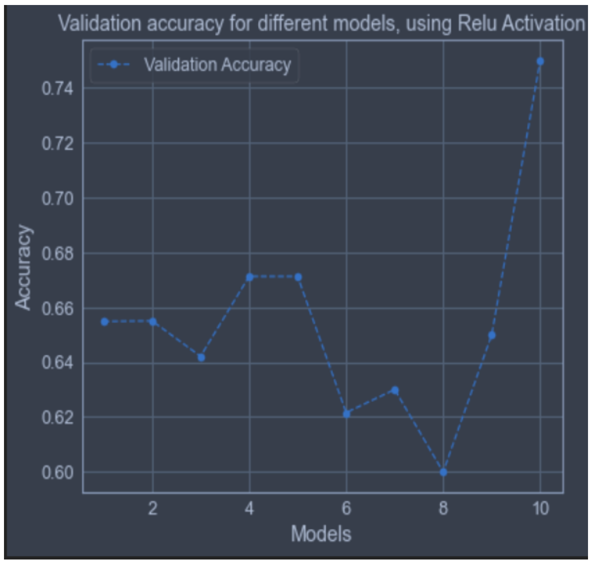
(Please Note that all the models trained and their results are attached as appendix A, at the end of this report. Here, only a few important models are discussed.)

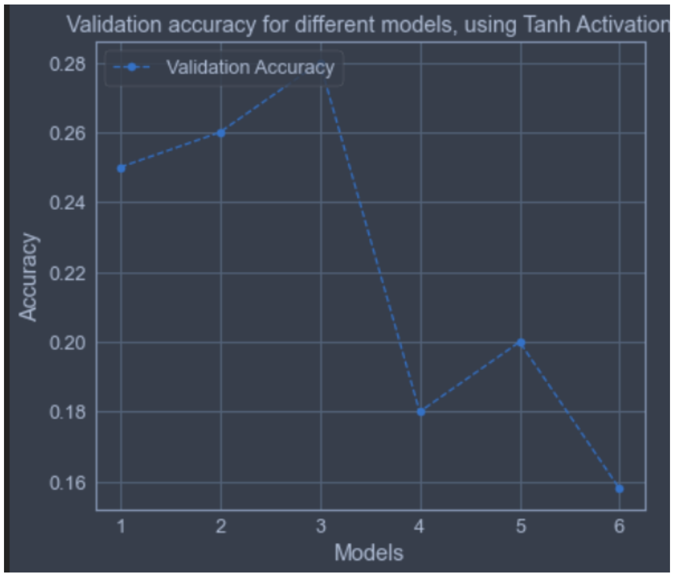
Following the reference paper model as our initial guess, in order to determine the correct activation function, we tried different models of the same architecture but different activations. Based on the obtained results, we conclude that tanh performs poorest, (Accuracy around 20%), sigmoid gives accuracy in the range of 30-40 %, while "Relu" acts the best giving validation accuracy around 60 %. Hence, for the subsequent models, we use Relu as our activation function. The reason for the same is as follows:
ReLU (Rectified Linear Unit) is a widely used activation function in deep learning due to its many advantages over the sigmoid and tanh functions:

- The ReLU function is significantly easier to compute than sigmoid and tanh, as it only requires a basic threshold operation on the input. This makes it faster and more computationally efficient, which is particularly essential for multilayered deep neural networks.
- Avoids the Vanishing Gradient Problem. This can make it challenging to train neural networks with multiple layers. The gradient of ReLU is constant for positive input values, so it does not suffer from this issue.
- RELU can minimise the network's computational complexity and enhance its generalisation performance by preventing overfitting. In practise, ReLU has been observed to converge significantly quicker than sigmoid and tanh, particularly when used with deep neural networks. This is because ReLU can preserve and propagate information across multiple layers, thereby enhancing the network's precision.

  In summary, ReLU is superior to sigmoid and tanh as a deep learning activation function because it is computationally efficient, avoids the vanishing gradient problem, has a sparsity property, and leads to quicker convergence.



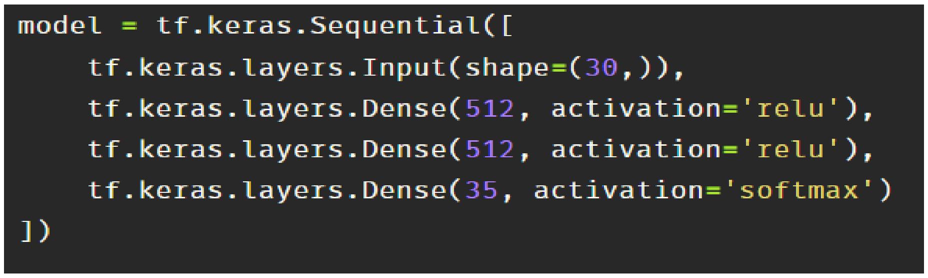


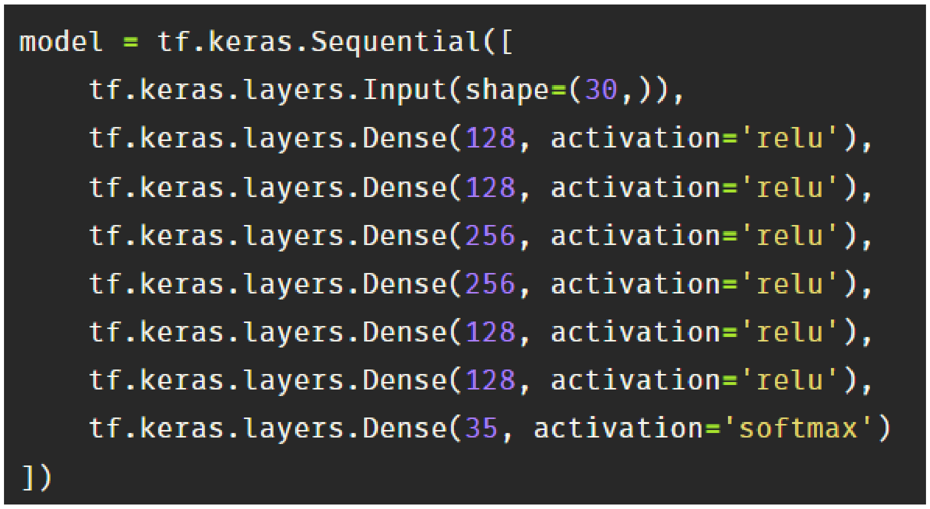


Activation Function	Accuracy Range
Relu	60% - 75%
Tanh	16% - 28%
Sigmoid	33% - 40%

Since Feed Forward Neural Networks are used, and in order to train models quickly consuming low resources, the number of epochs are kept as 10. Since the architectures trained are to be kept as low memory as possible, more preference was given to training models with lower number of nodes. Further, at a time only 1000 batches of input are fed to the network for training,(About 1/3rd of training data) to keep RAM resources as low as possible, and to avoid overfit & bias by changing data continuously for training. Further in order to avoid overfit, we use the early stop functionality of Tensorflow, whereby we monitor the validation loss, using the patience = 3, i.e. if the validation loss does not improve after 3 epochs, the training will be stopped early.

Now, the best accuracy around 75 % was reported in two of the following architecture,


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```



```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

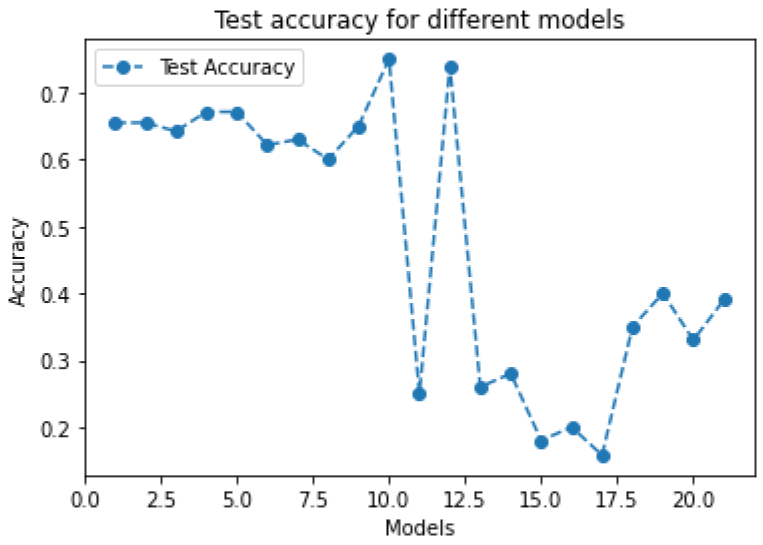
The effect of modifying the weights or layers in a neural network design on the accuracy of the model was varying depending on a number of factors. It includes the extent of the modifications, the complexity of the model, and the nature of the data used. Small changes to the weights or layers of a neural network had a comparatively minor effect on the accuracy of the model, whereas larger changes had a greater effect. Changing the weights minimally, for instance, results in a small decrease in accuracy, whereas removing a layer or adding a large number of new neurons had a substantial decrease in accuracy. In addition, the architecture of the neural network, the number of layers, the size of the training dataset, and the training learning rate also influenced the impact on the accuracy. In certain cases, adding more layers or neurons to a neural network results in overfitting and a loss of accuracy. We see that for RELU activation, maximum of the accuracies are in the range (60%,75%) . Even when decreasing the model size to 2 hidden layers of size 128, we get the accuracy of 63%, which is almost in the same range and giving a vast advantage since the number of neurons are used less along with the number of layers in this case. Further, increasing the number of neurons in the first hidden layer from 32 to 64 to 128 increases the accuracy, and the convergence is quick to be seen in this case. As seen between models 3 & 4,as we add additional hidden layers of size 64 at start and end, the accuracy decreases from 75% to 68%. Thus, upto a certain point, increasing the size of the model is beneficial. This can be due to the fact that the neural network is able to extract more information or more features in the first layer from the input data, and these features help in subsequent processes to classify the output in a better way, & to increase the variance between the features of different labels. Further, the change in the middle hidden layers of the neural network has a small effect on the output accuracy. As seen, addition of an extra hidden layer of size 256 in model 2 gives almost the same accuracy as seen in model 1. Thus, the changes in the first hidden layer and penultimate hidden layers has a greater toll on accuracy than changing the middle hidden layers.

Both of the optimal models can be used, and they have their own advantages. The first model has fewer layers, and hence it can train and give results quickly. However, in the first case, we need a matrix of larger size (512,512,n) for computations. On the other hand, the 2nd architecture has more layers than the previous model, but the matrices generated are relatively small. Hence, in cases where the device can't allocate huge memory for computations, the 2nd architecture works better.

On decreasing the model size, in general, the accuracy of the models tend to decrease. This may be due to the fact that data ought to be underfitting in these cases.

Number of Layers	Validation Accuracy	Validation Loss
2	63%-66%	3.657
3	70%-73%	0.85
4	65%-68%	5.406
5	66%-70%	7.895
6	66%-68%	13.113
7	73-75%	0.4558

Summary of our models :



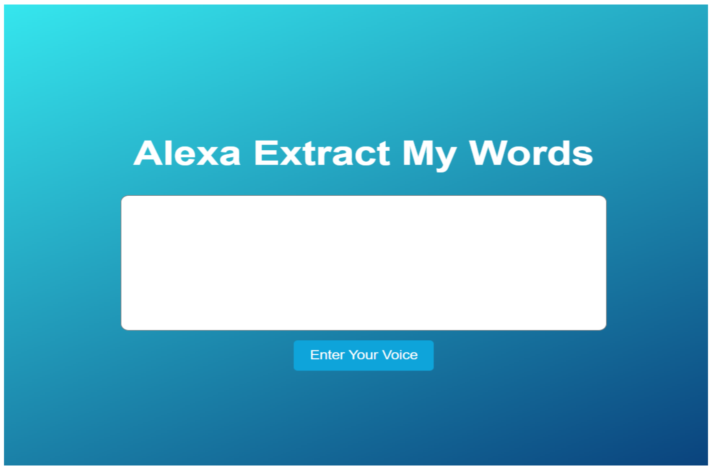
The different models trained and their test accuracy is attached. The maximum test accuracy was found to be around 75% for following two architectures.

```python
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

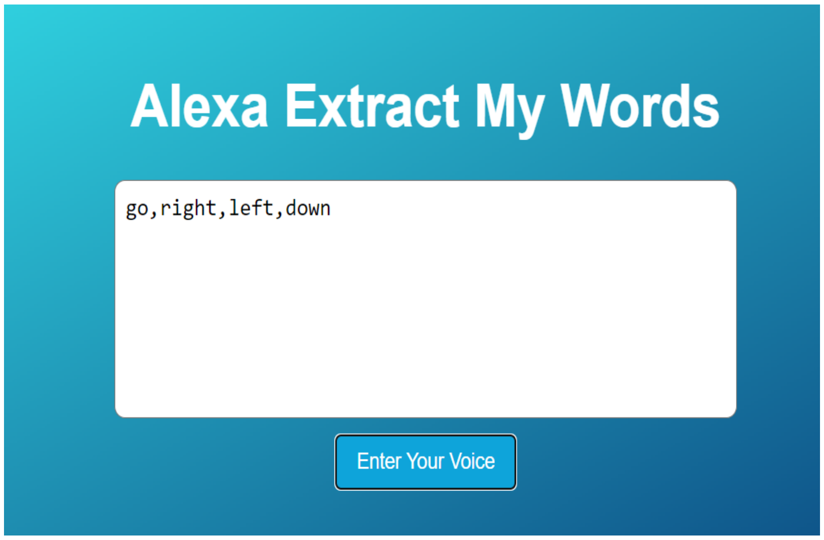
```python
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

Deployment :

We deployed our Application by loading our best model. The user would be providing his/her audio. The model would be accepting the audio and would recognise the main keywords present in the statement. At the end it would display these keywords on the screen. The UI interface is being provided below.



The user can start adding his voice by clicking on the “Enter Your Voice” Button. The application would display the keywords that were extracted by our model and would display the same on the screen. The demo of the same is provided below.



The User provided with the statement “Go Left And Right And Down The Hill”. The model extract Go, Right, Left and Down and displayed the same onto the screen.
These was the Application that was deployed.

Contributions :-
Together, we perform the majority of the task, making it extremely difficult to distinguish between our contributions.

Mistry Parth Kirsankumar (B20EE090) :-
Data Preprocessing, Data Visualization, Pondering different Deep Learning Architectures, Finding Best Model,Deployment of Application, Report Making

Nagda Yash Narendra (B20EE092) :-
Data Loading, Implementation of Deep Neural Networks, Observations and Deducing Inferences, Report Making

References:

- Google Speech Dataset:
  https://paperswithcode.com/task/keyword-spotting
  https://arxiv.org/pdf/1804.03209v1.pdf
  R. Tang and J. Lin, "Deep Residual Learning for Small-Footprint Keyword Spotting," ArXiv e-prints, Oct. 2017.

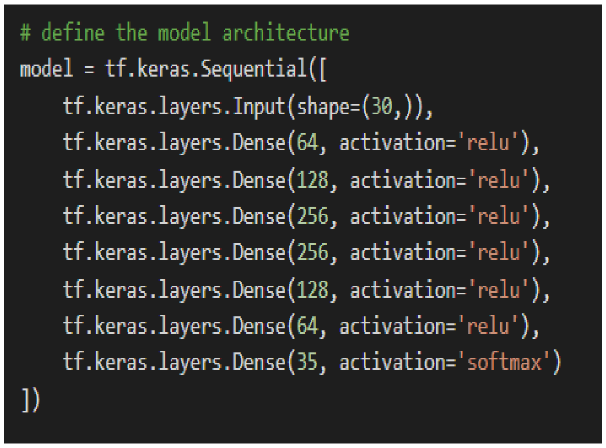
- MFCC Reference:
  https://medium.com/@tanveer9812/mfccs-made-easy-7ef383006040
  https://en.wikipedia.org/wiki/Mel-frequency_cepstrum#:~:text=Mel%2Dfrequency%20cepstral%20coefficients%20(MFCCs,%2Da%2Dspectrum%22).

- Dataset Can be downloaded from following Link:
  http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz

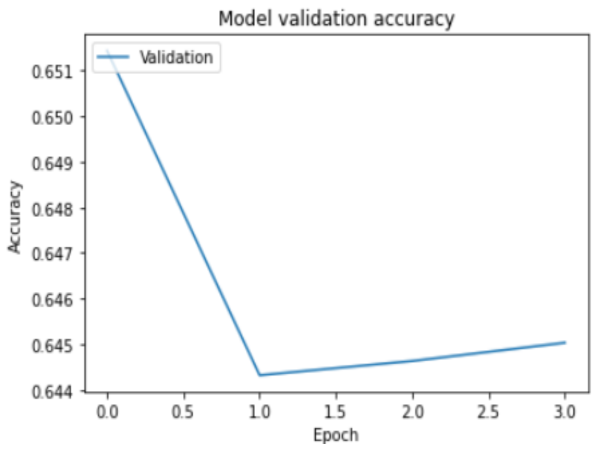
- Feed Forward Neural Networks
  https://www.researchgate.net/publication/228394623_A_brief_review_of_feed-forward_neural_networks
  https://ieeexplore.ieee.org/document/329294

Appendix A) :

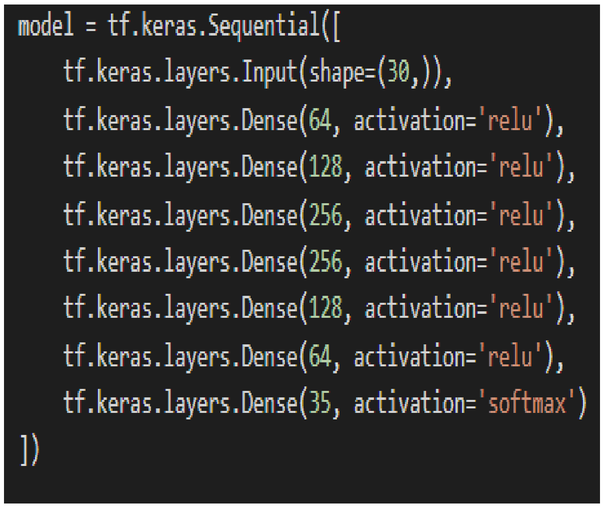
1] Model1 :-



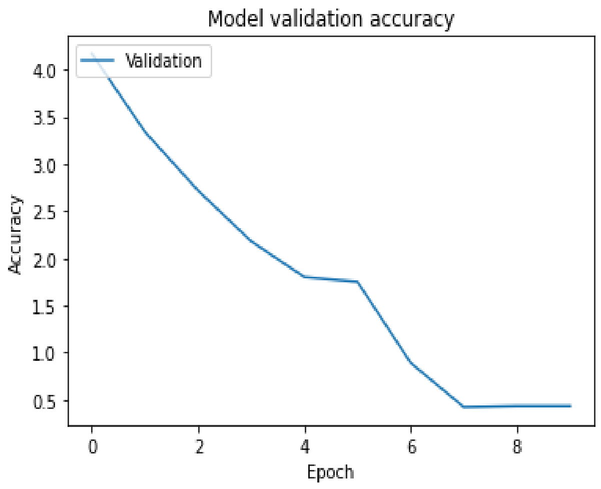
```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```



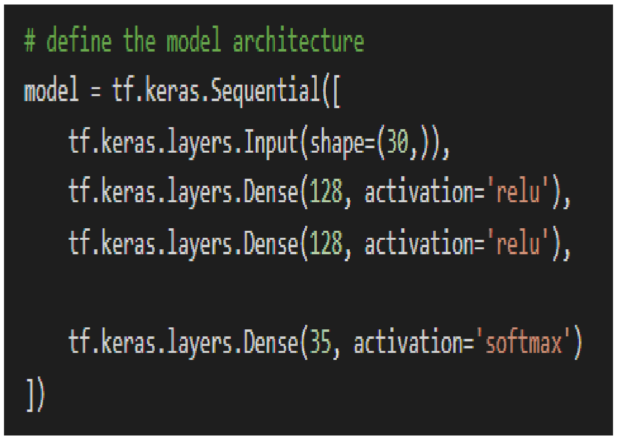
2] Model2 :-



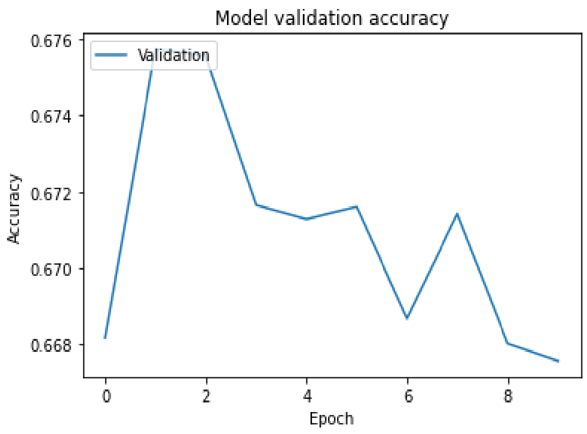
```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```



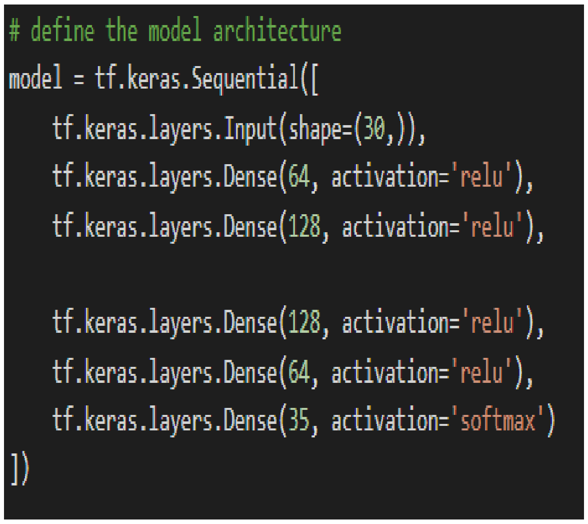
3] Model3 :-



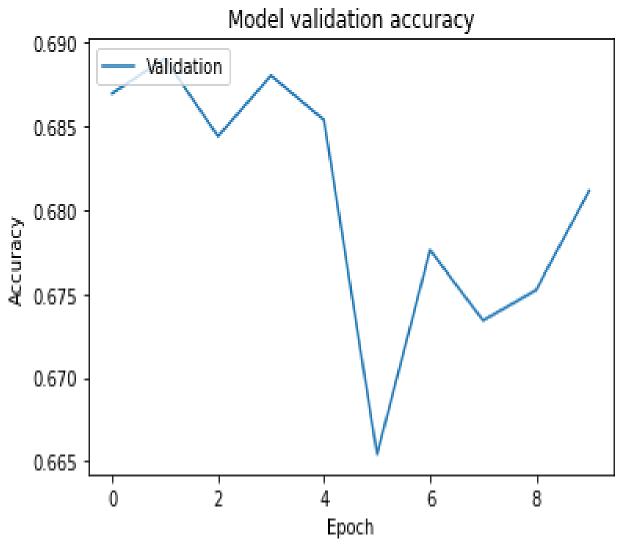
```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),

    tf.keras.layers.Dense(35, activation='softmax')
])
```



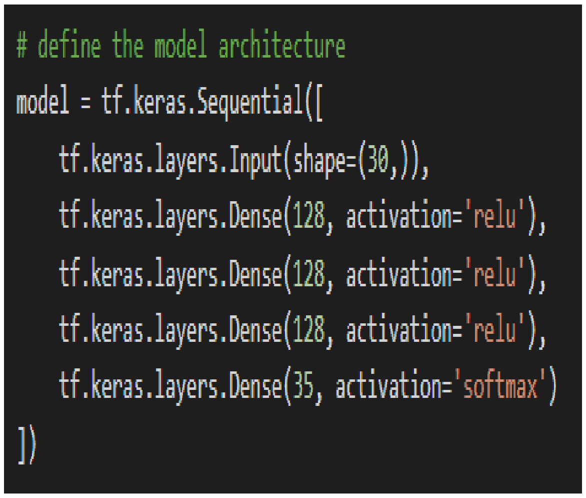
4] Model4 :-



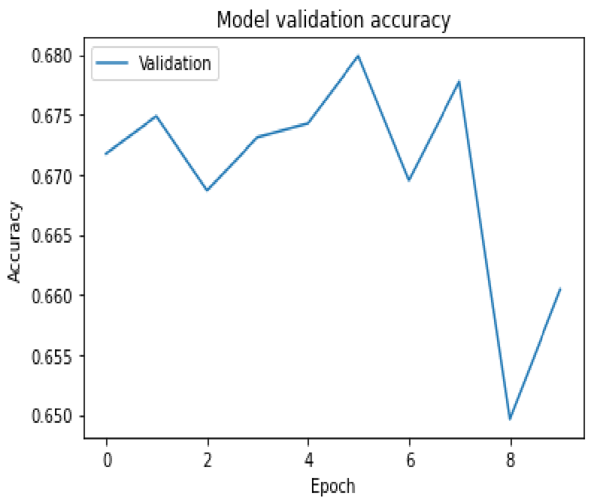
```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```



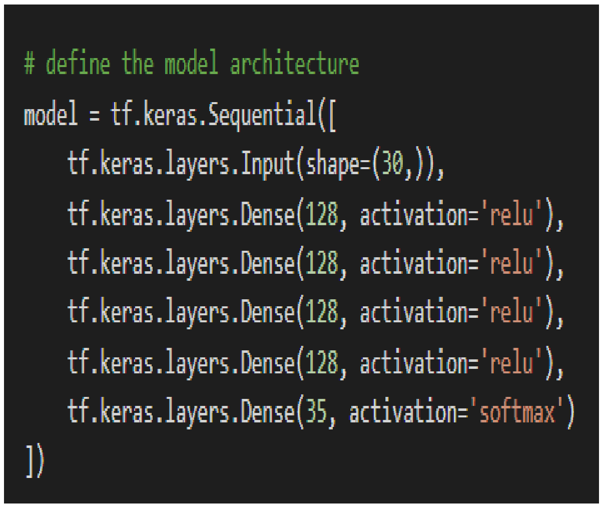
5] Model5 :-



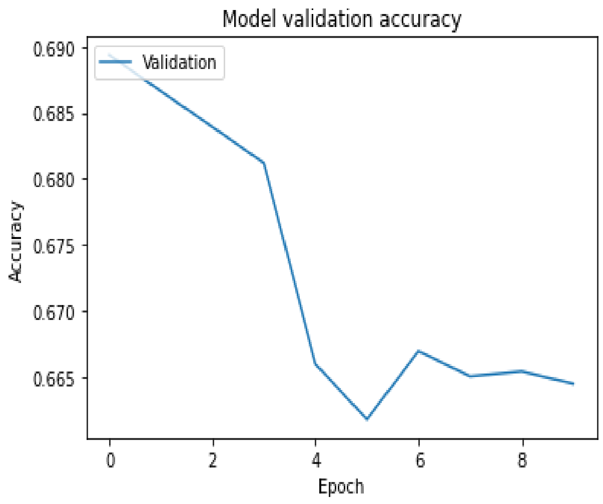
```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```



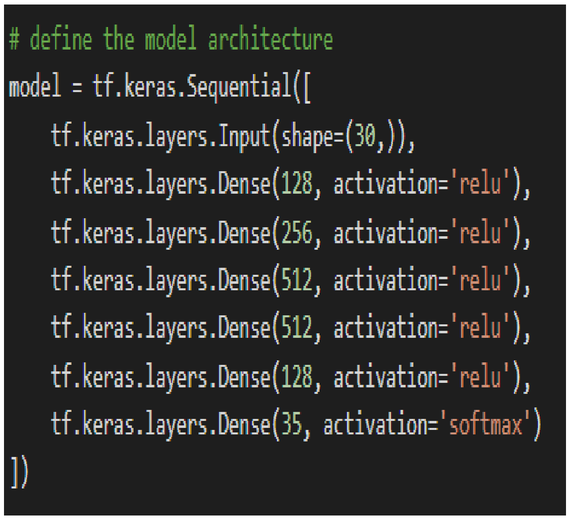
6] Model6 :-



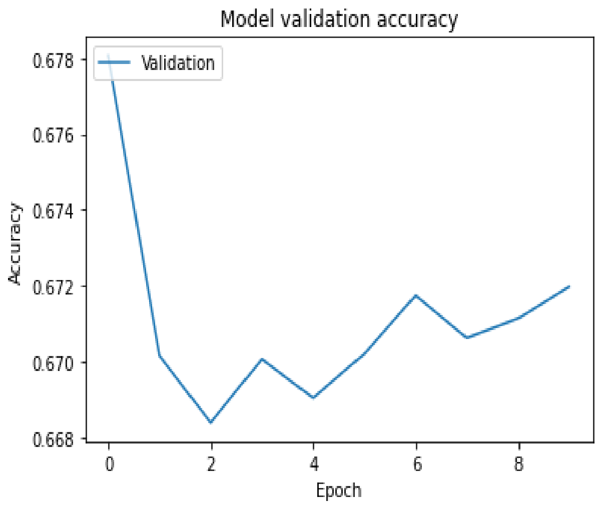
```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```



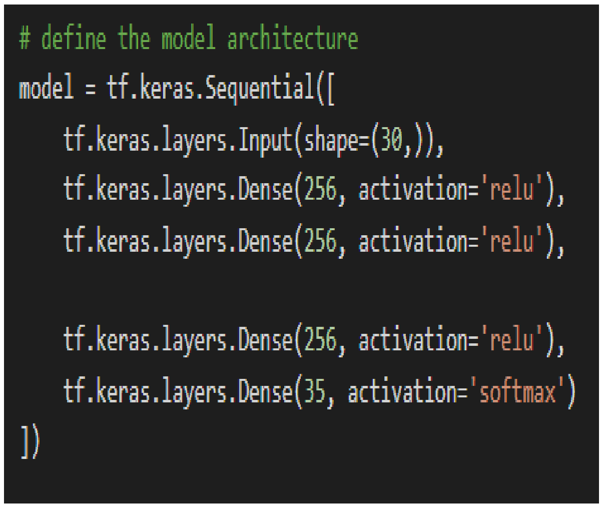
7] Model7 :-

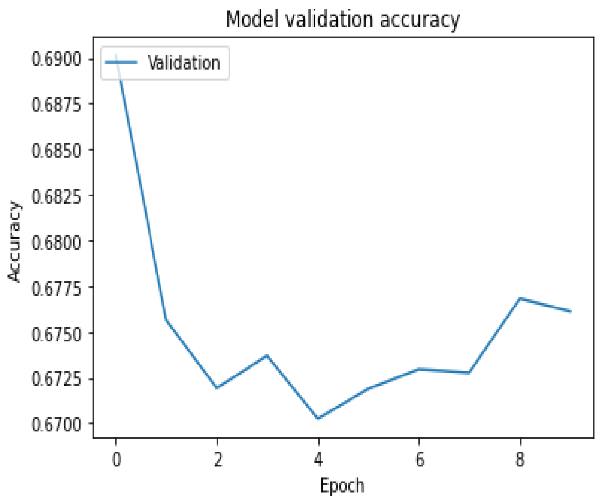


```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

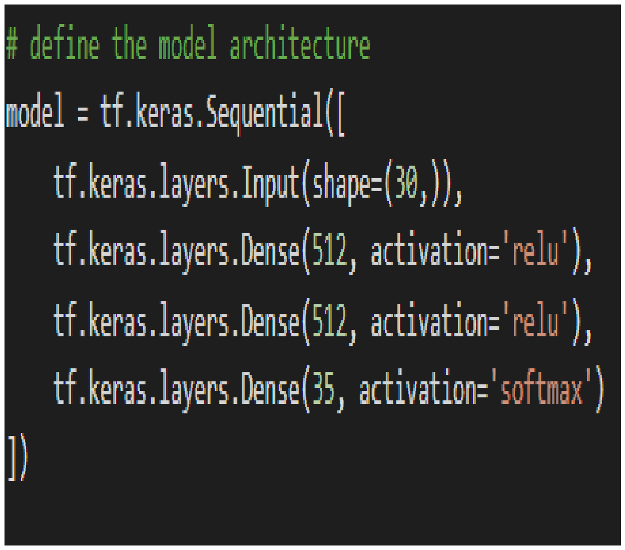


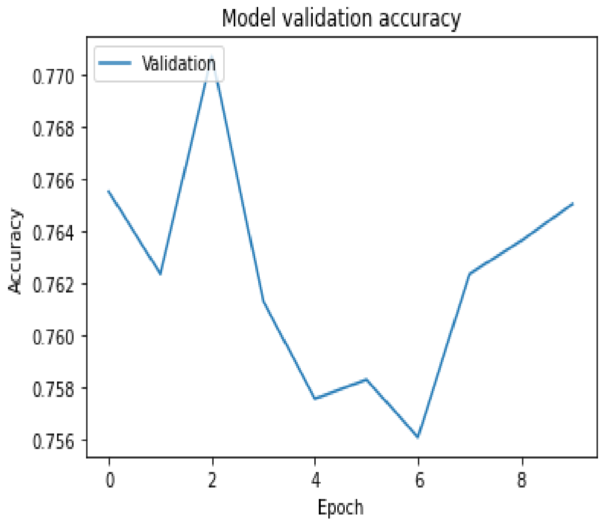
8] Model8 :-


```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

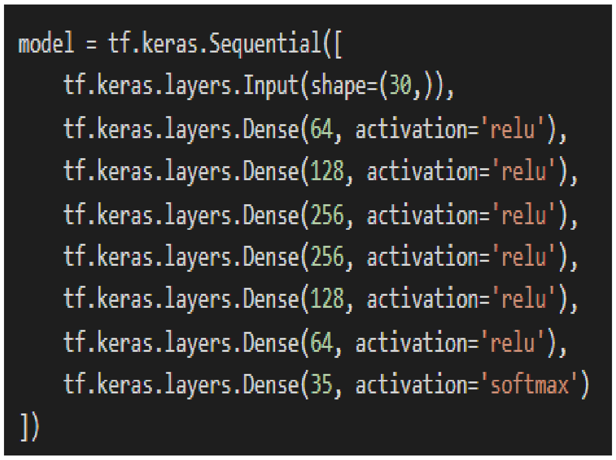


9] Model9 :-

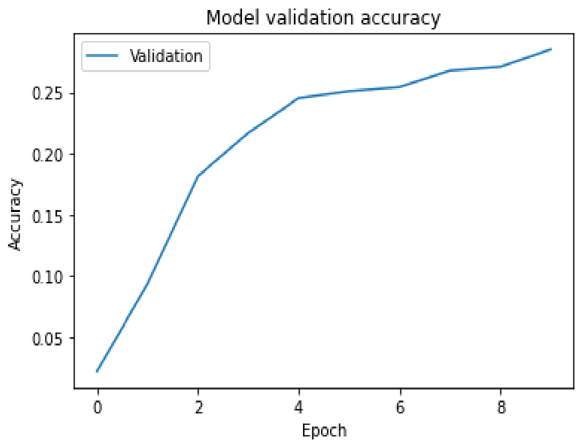

```
# define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

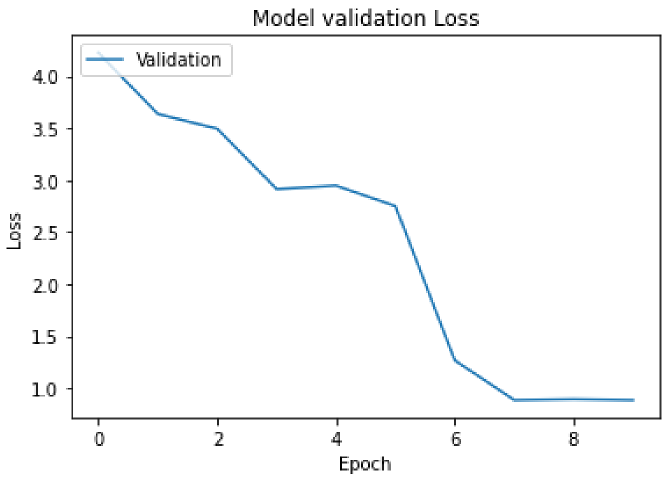


10] Model10 :-

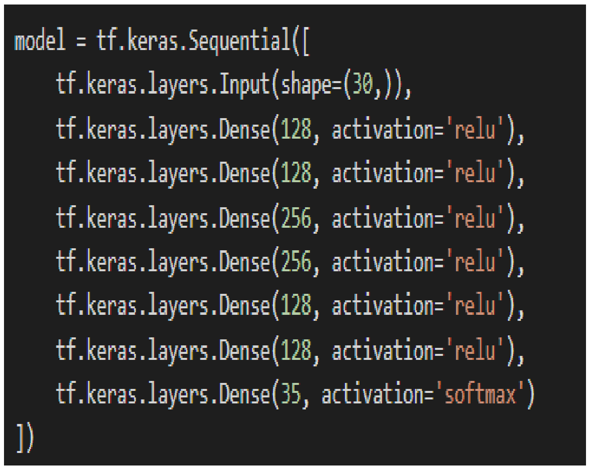


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

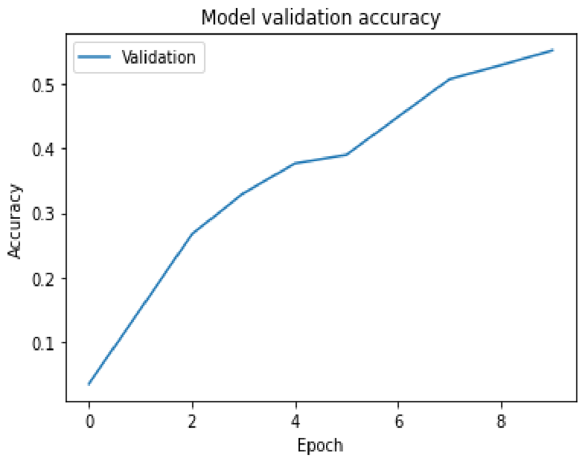


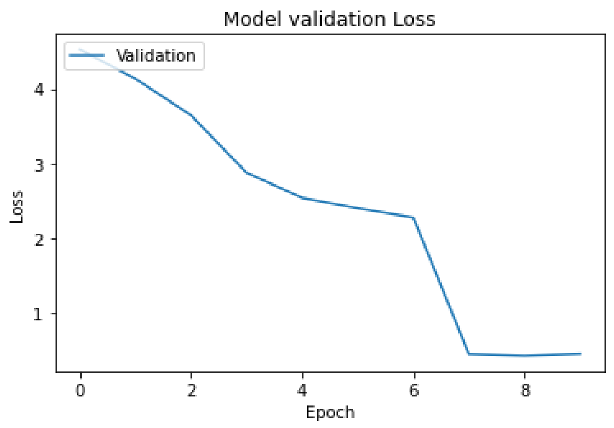


11] Model11:

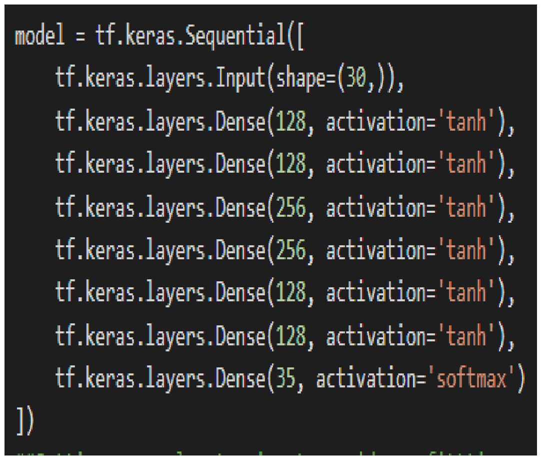


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

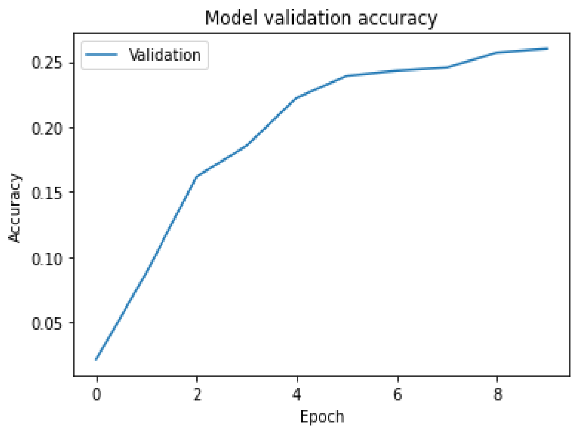


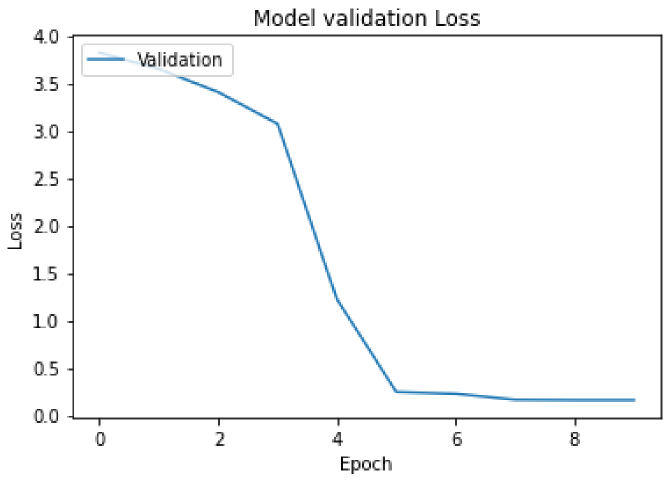


12] Model12:

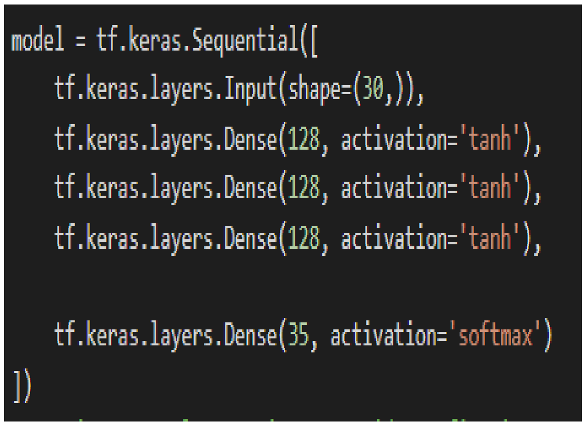


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(256, activation='tanh'),
    tf.keras.layers.Dense(256, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

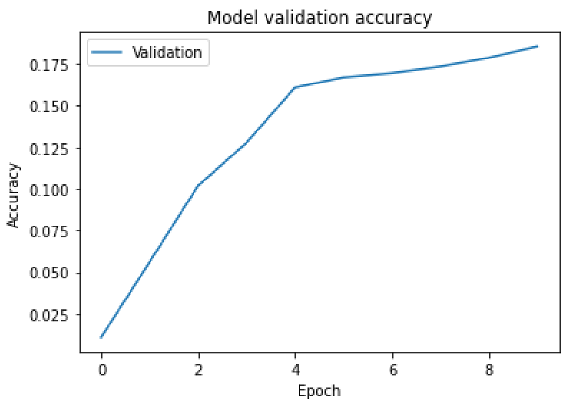


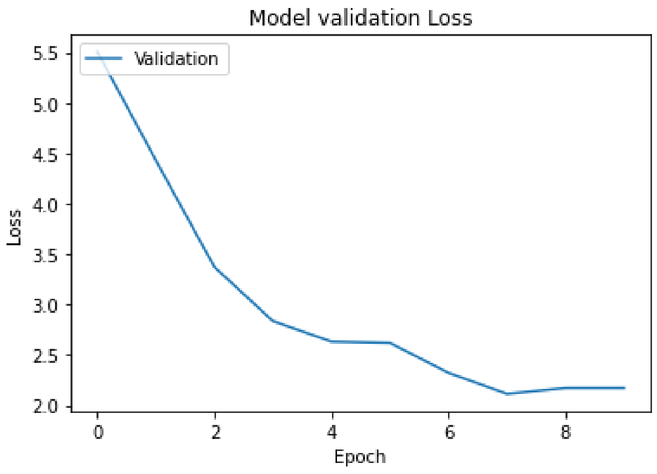


13] Model13:

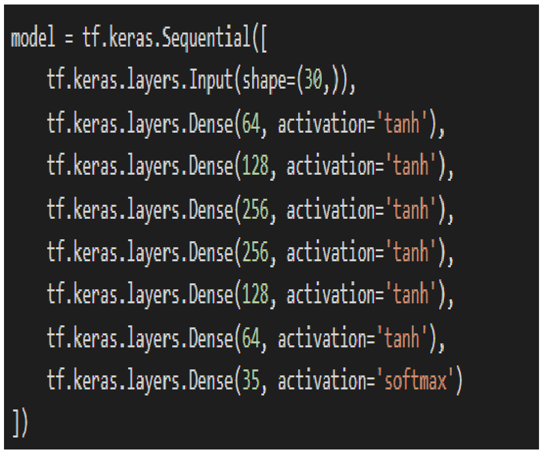


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),

    tf.keras.layers.Dense(35, activation='softmax')
])
```

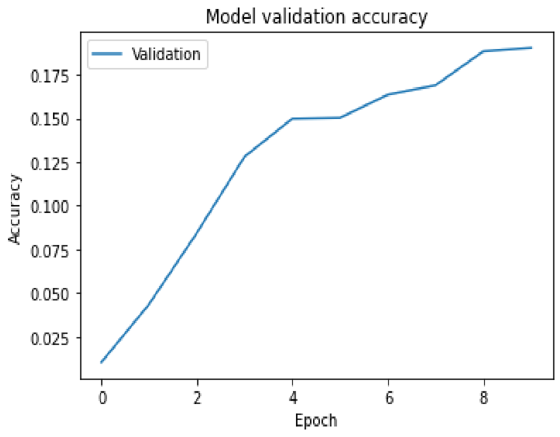


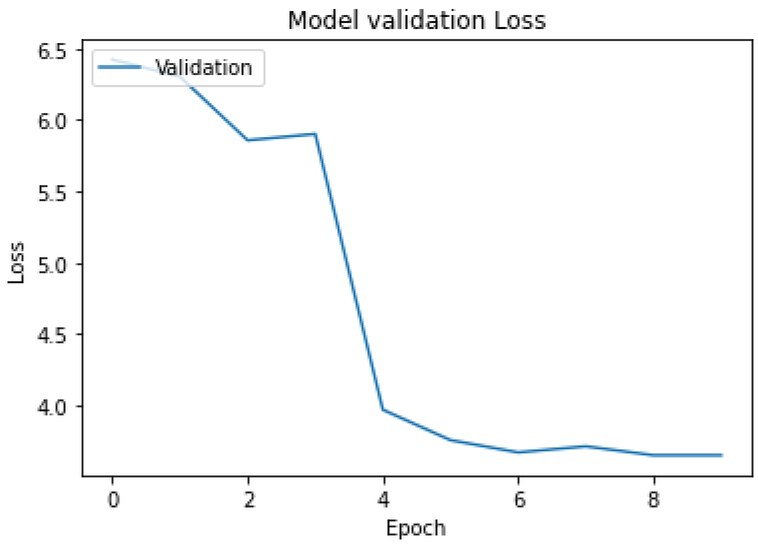


14] Model14:

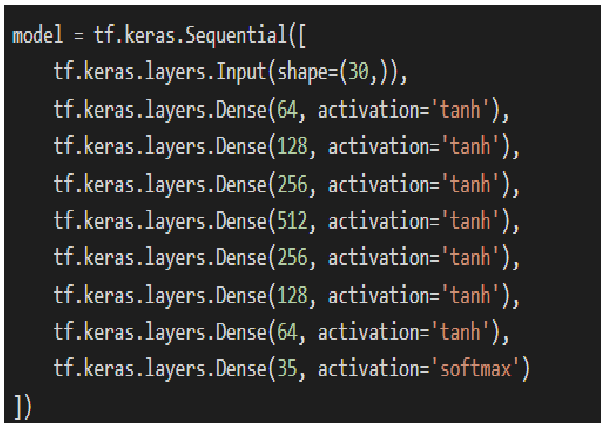


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(256, activation='tanh'),
    tf.keras.layers.Dense(256, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(64, activation='tanh'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

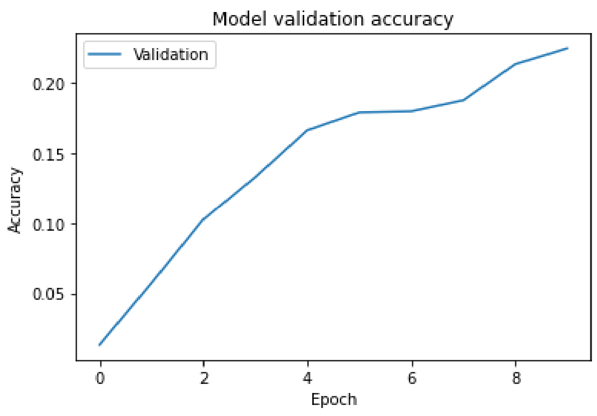


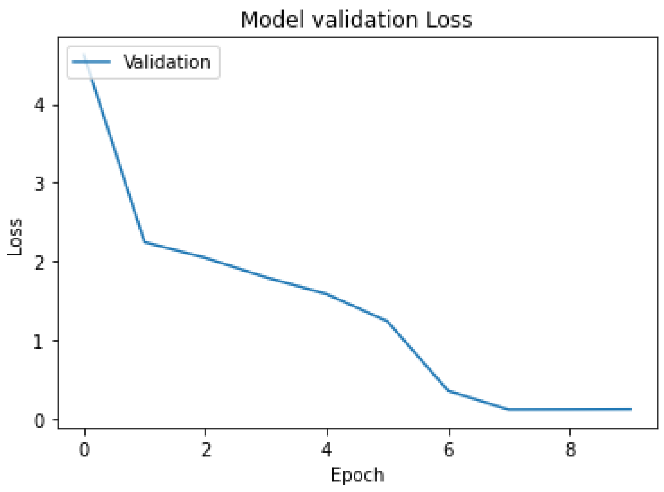


15] Model15:

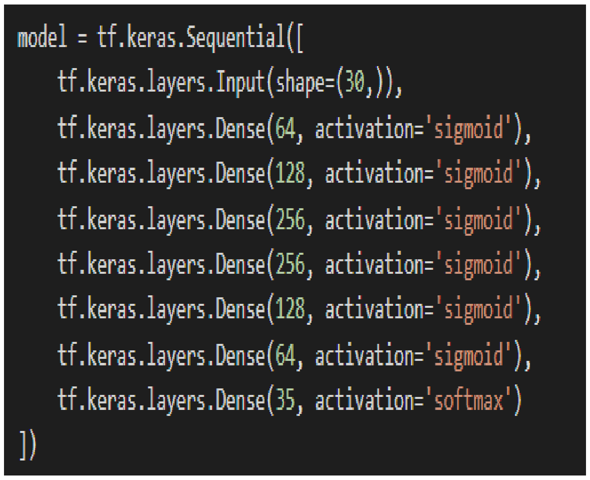


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(256, activation='tanh'),
    tf.keras.layers.Dense(512, activation='tanh'),
    tf.keras.layers.Dense(256, activation='tanh'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(64, activation='tanh'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

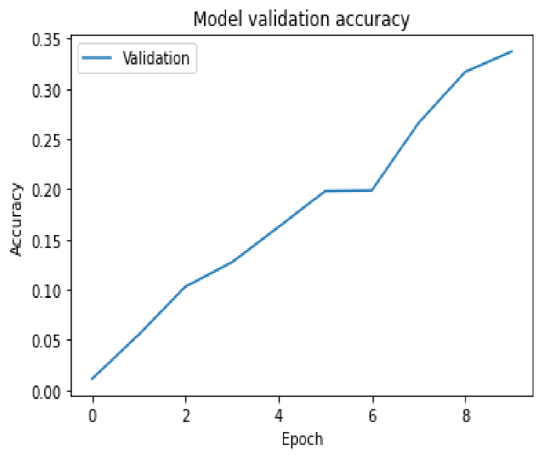


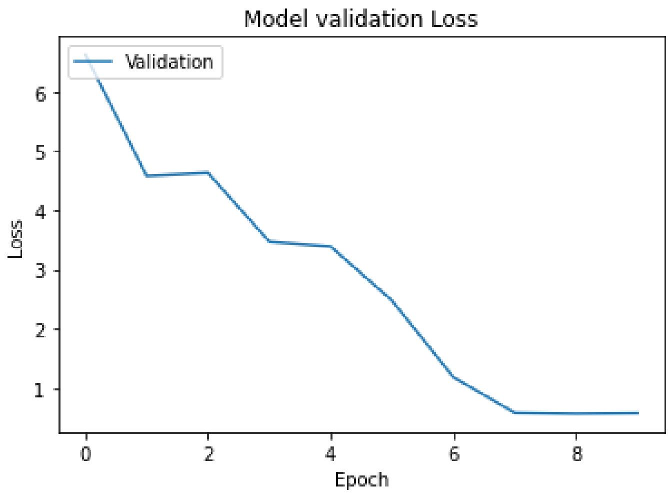


16] Model16:

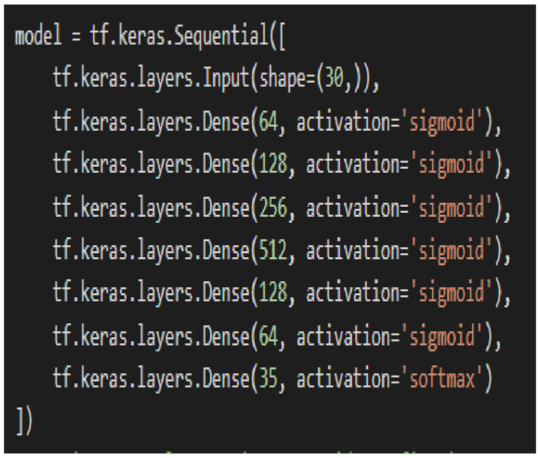


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(256, activation='sigmoid'),
    tf.keras.layers.Dense(256, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(64, activation='sigmoid'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

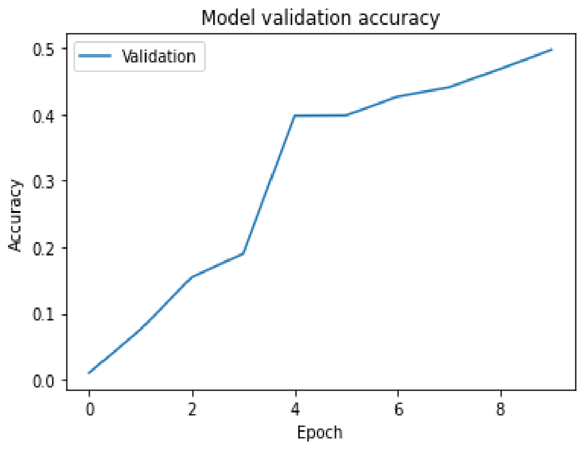


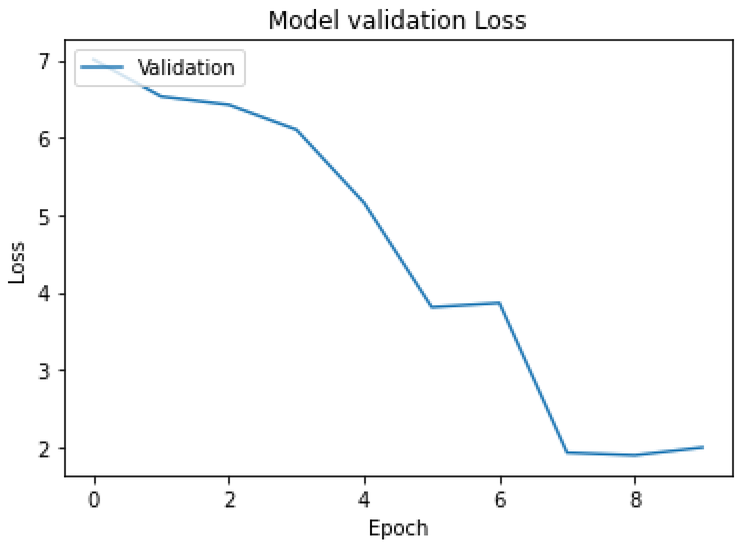


17] Model17:

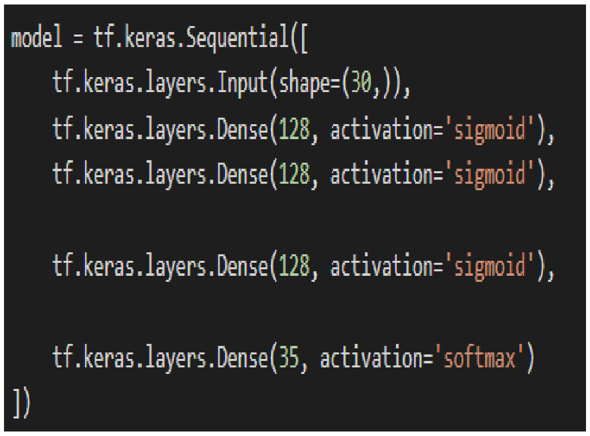


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(64, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(256, activation='sigmoid'),
    tf.keras.layers.Dense(512, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(64, activation='sigmoid'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```

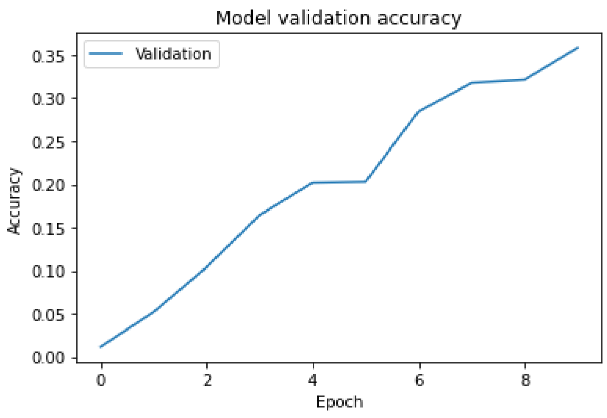


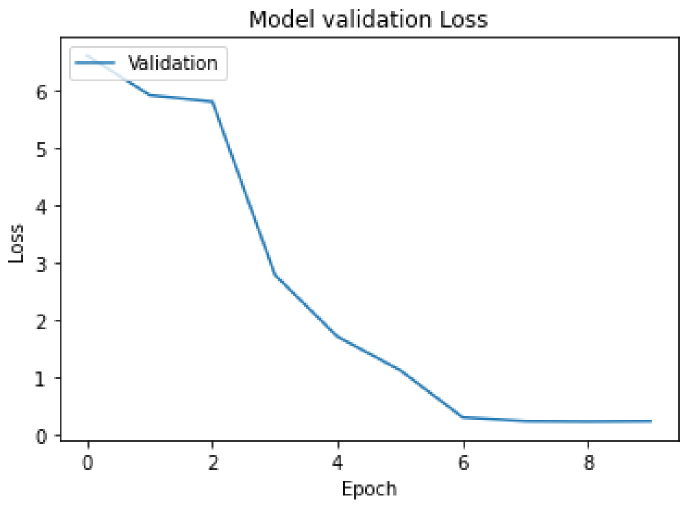


18] Model18:

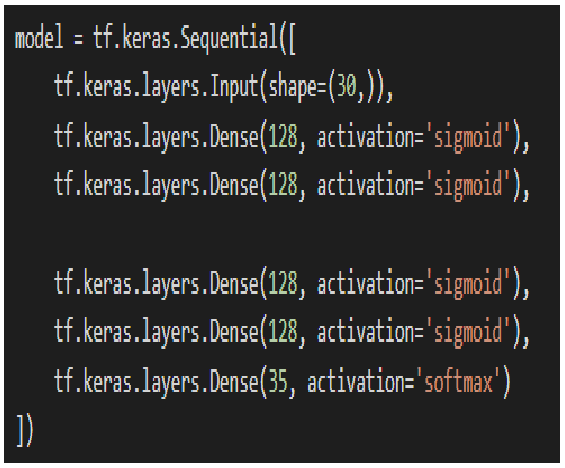


```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),

    tf.keras.layers.Dense(128, activation='sigmoid'),

    tf.keras.layers.Dense(35, activation='softmax')
])
```

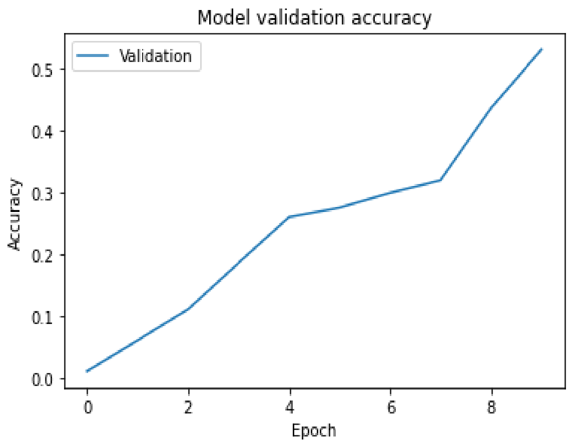


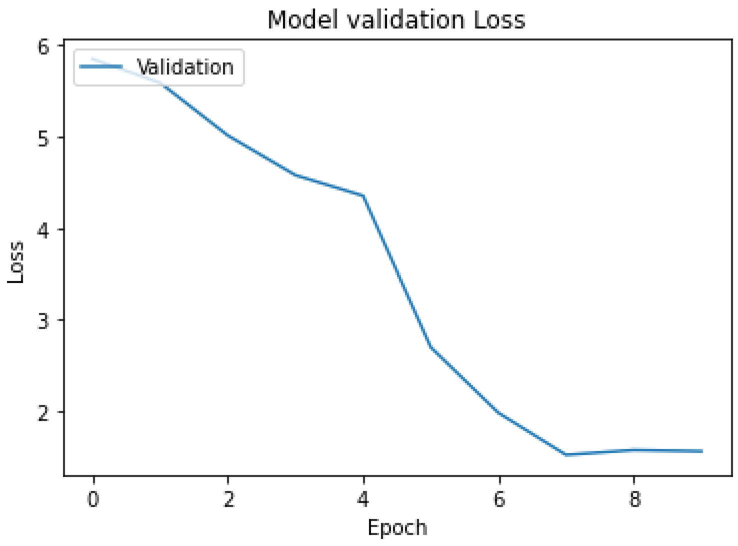


19] Model19:



```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(30,)),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),

    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(35, activation='softmax')
])
```


Model validation accuracy
Validation
Accuracy
Epoch


Model validation Loss
Validation
Loss
Epoch