# ON A NON-NOVEL DEEP LEARNING CLASSIFICATION OF AN UNKNOWN IMAGE DATASET.

**Jimmy Tsz Ming Yue**
School of Computer Science
University of Sydney
jyue6728@uni.sydney.edu.au
SID : 440159151

**Melissa Tan**
School of Computer Science
University of Sydney
mtan2988@uni.sydney.edu.au
SID : 200249191

**Zhuoyang Li**
School of Computer Science
University of Sydney
zhli2344@uni.sydney.edu.au
SID : 480164337

May 31, 2019

## 1 Introduction

### 1.1 Overview and Motivations

The ability of machine learning structures to classify images have widely expanded since the introduction of deep learning architectures, in particular Convolution Neural Networks [1]. Such networks are wide and varied in their configuration and construction, with a remarkable amount of literature [2] detailing different convolution stacks along with methods such as residual networking that increase performance within deeper structures [3]. In this vein, our study seeks to compare and contrast a variety of non-novel deep learning configurations through the construction of suitable transfer learning networks in order to complete the computer vision task of classifying an unknown image dataset. Through this examination, the influences of such differing configurations and their ability to generate highly performing classifications can be determined. By obtaining an understanding of the theoretical nuances within each individual configuration, which is presented in our methodology, and its relative levels of classification performances, we are able to consider the efficacy of such learning architectures in future studies especially that related to computer vision.

### 1.2 Dataset

The given train dataset consists of 31,925 images and 15,516 test images along with a set of 20 unique labels in which each image may belong to one or multiple labels. In the train dataset, approximately 77.96% belongs to only one label.

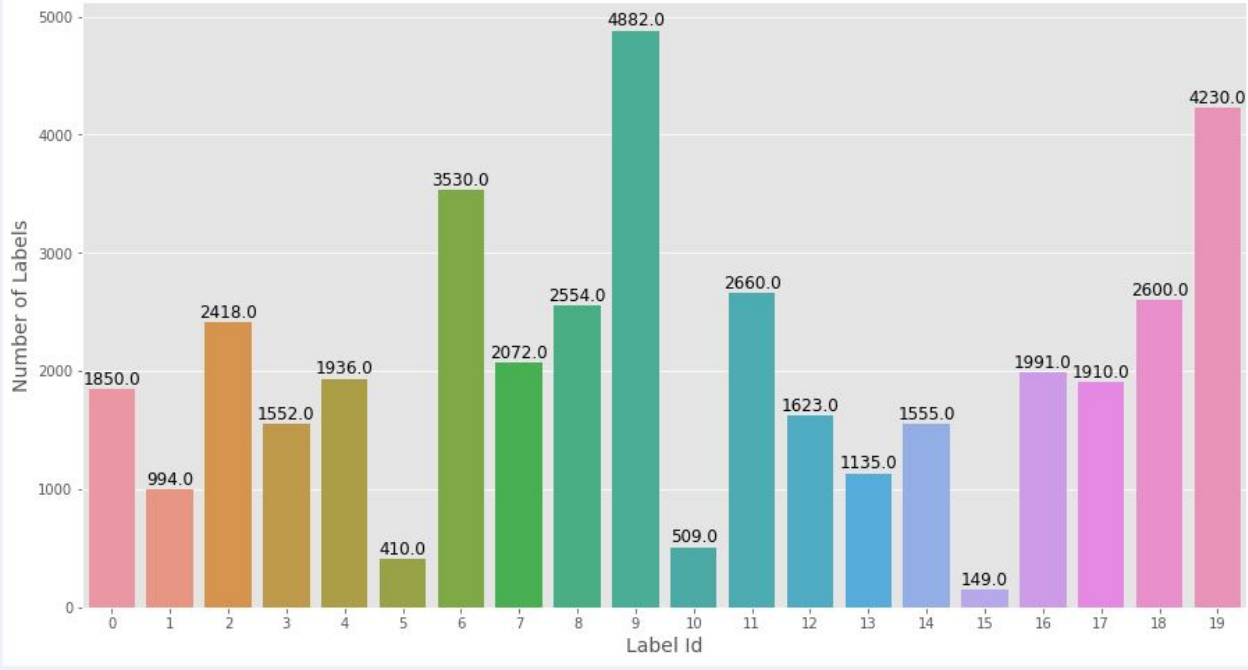


***Figure 1:*** *Distribution of Class Labels within Unknown Dataset*

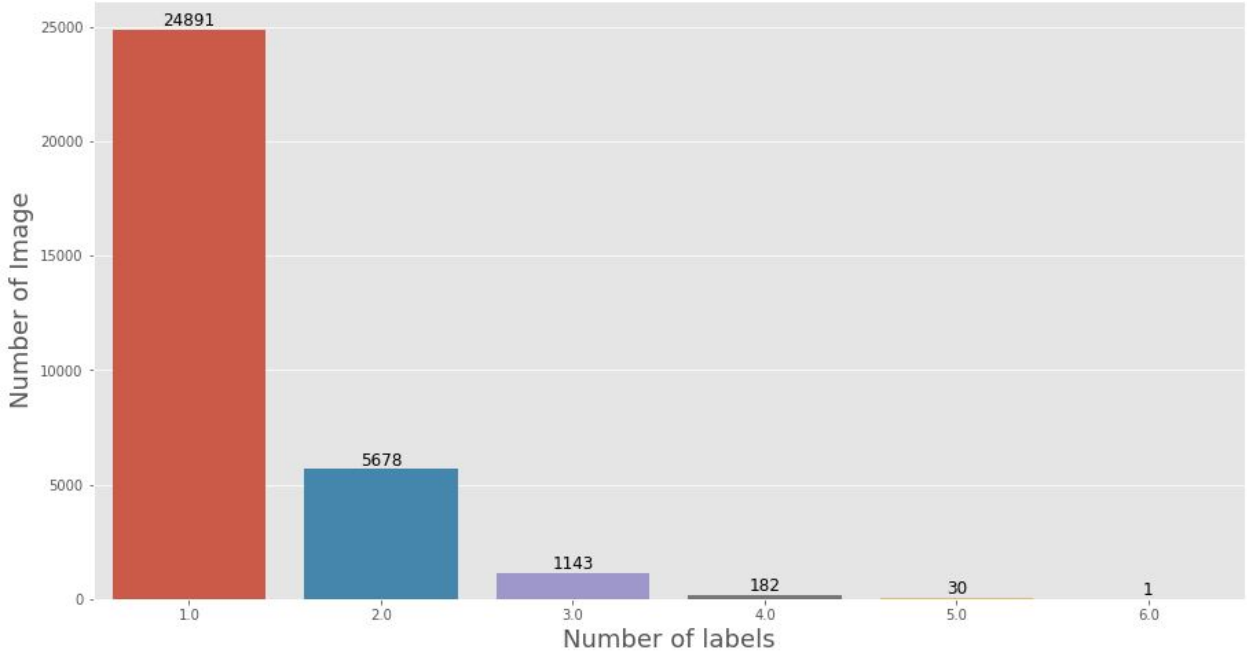


***Figure 2:*** *Distribution of images possessing multiple class labels.*

## 2 Methodology

### 2.1 Preprocessing

In generating a varied data-set for a Computer Vision task, the robust ability of convolution Neural Networks (ConvNet) to accurately generate relative classification performance in spatial invariance allows for the expansion of small and

limited datasets [4] .Notably, ConvNet invariance to spatial shifts such as translation, size, and illumination is utilised in this study to expand our given image dataset as a hope of improving generalisation. Firstly the dataset was rescaled by a factor of $1/255$ A rotation range of $40$ degrees was selected along with randomised shearing, horizontal and vertical shifting at ranges of $0.2$, $0.1$ and $0.1$ respectively. Furthermore Images are allowed to be horizontally flipped as a means of variation. This dataset augmentation under such spatial modifications were then split in terms of training and validation to monitor classification within our given task domain.

### 2.2 Deep Learning Architecture

Since the proposition of Artificial Neural Networks in [5] as an emulation of a biological computational framework yields remarkable results in the ability of statistical modelling. Like this biological analogue, computational units aptly labelled "Neurons" are organised in group like "layers" such that the Networks generated are able to handle vectorised computational methods in an efficient manner. Such layered organisations are mathematically formalised in the form of Graph structures $G(V, E)$ such that the Neurons are represented in the form of vertices $V$ and weighted directed edges $E$ are fed into the inputs of our computational neurons. Starting from an input $V_0$ layer, the flow of information propagates throughout the network through each layer $V_i$ reaching an output layer, where the vertices $v_{i,j}$ refers to the $j$-th neuron with the $i$-th layer. Activation functions of the inputs of such neurons are generally non-linear [6], with the Rectified Linear Unit (ReLu) being common within most modern Deep Learning structures due to its biological similarity [7] and its remarkable computational efficiency [7]. Such Networks are trained through the optimisation of the weighted edges with regards to a specific output. Generally, these optimisation procedures presents the introduction of a loss function that describes the mathematical distance (under some metric) between the desired output from an output generated through some combination of weights. Through the minimisation of this loss by method of back-propagation an optimised Network can be produced. Utilising such structures, a variety of tasks can be conducted such as the computer vision task of generating classification labels of images through the usage of additional deep learning methods such as that of convolution.

### 2.3 ADAM optimisation

Adaptive Moment Estimation (ADAM) is an adaptive consideration of the method of momentum's in stochastic gradient descent. First proposed in [8], the adaptive methodology utilises the first moment along with the second moment of the gradients during optimisation. Despite maintaining a similar structure to that of the method of momenta [9] the learning rate is adaptive at each iteration of the gradient descent. This yields for a faster level of convergence for the optimisation. The mathematical representation of each step within the algorithm is reproduced as in [8] :

$$m^{(i+1)} := \beta_1 m^{(i)} + (1 - \beta_1) g^{(i)} L \tag{1}$$

$$v^{(i+1)} := \beta_2 v^{(i)} + (1 - \beta_2)(g^{(i)} L)^2 \tag{2}$$

$$\hat{m} = \frac{m^{(i+1)}}{1 - \beta_1^{(i+1)}} \tag{3}$$

$$\hat{v} = \frac{v^{(i+1)}}{1 - \beta_2^{(i+1)}} \tag{4}$$

$$w^{(i+1)} := w^{(i)} - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon} \tag{5}$$

where $m^{(i)}$ describes the estimated first moment with bias corrective factor $\hat{m}$ at the $i$-th step and $v^{(i)}$ the estimated second moment with bias corrective factor $\hat{v}$ at the $i$-th step. After each successive iteration, the weight $w$ is returned. Due to its rapid convergence rates along with its ability to escape gradient "ravines" [10] it is the standard gradient descent optimiser of most deep learning structures. Under such consideration, ADAM becomes our choice of optimisation within this study.

### 2.4 Dropout

As an architecture possess more and more layers, the likelihood of the network in over-fitting becomes increasingly more probable. Like most learning strategies there exists a variety of regularisations that seek to limit such an over-fitting effect. One such regularisation is the implementation of dropout throughout a given structure. First introduced by Srivastava [11], the principles behind dropout considers the training of ensembles consisting of sub-networks that are spanned through the removal of non-output nodes. Through this consideration, these ensembles are prevented from co-adapting.

### 2.5 Batch Normalisation

Within deeper Network structures, the increase of layer depth corresponds to a difficulty of algorithmic convergence within gradient descent methods. This difficulty arises due to the changes in data distribution within each layer as the refinement of previous layers causes a shift in parameters. This results in a reduction in the training speed as lower learning rates are required to compensate. As such Batch Normalisation (BN), proposed in [12] suggest the use of BN in choosing an aggresive learning rate. This is done through how BN continuously updates the intermediate output of mini-batch SGD architectures within each iterative step by normalising the batch $\mathbb{B}$ means and variances. At each activation of layer $V_i$ of a constructed network, BN transforms the input $x$ through a normalisng function:

$$f(x) = \alpha \left( \frac{x - \hat{\mu}}{\hat{\sigma}} \right) + \beta \tag{6}$$

where $\hat{\mu}$ and $\hat{\sigma}$ are the estimates of the mean and standard deviation within batch $\mathbb{B}$ and $\alpha$ and $\beta$ are coefficients considering scaling and offset. In doing so, this generates a rescaling effect for the input data for each layer, and shifts the data to be normally distributed. In doing so, this allows for the selection of a high learning rate without the divergence of the algorithm. As an additive effect, noise generated from our mean and variance estimates within batching provides a regularising effect that seeks to prevent over fitting from larger and larger network depths.

### 2.6 Convolution and Convolution Architectures

As mentioned above, computer vision tasks generally employ the convolution as a means of dealing with structures that exists within images. In order to understand this we present the theoretical preliminaries of convolutions as follows: Convolution refers to the mathematical operator in which two functions; the input $x(t)$ is combined with a kernel $K(t)$ under the convolution operation $*$;

$$s(t) = (x * K)(t) \tag{7}$$

which generates an output or feature map $s(t)$. Specifically in our case of computer vision in generating labels for images [1], a two-dimensional image $I$ is considered as our input $x$ with kernel $K$ producing the feature map;

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n). \tag{8}$$

This is the crux to CNN systems and serves as the activation function $\sigma$ for neurons in the early layers of the neural network. Under different convolution layers, a filter is generated which effectively extracts features under a small patch of the image for consideration. Generally such features are related to the edges or colours of the images [1], and such layers can be arranged successively to generate a cascading effect, allowing for multiple features to be extracted in a manner of heightening the learning of deep networks.

### 2.7 Pooling Layers

Within most convolution architectures, after subsequent layers of convolution and rectified linearity within basic dense structures, generally a process called pooling is used to modify the output of convolution. There are many methods and types of pooling that is conducted, however in our study the one we will implement is that of the MaxPool which seeks to report the maximum output within a rectangular neighbourhood [13]. This is done through the reduction of spatial volume passing through further layers. The pooling layers takes an input of size:

$$i_{\text{size}} = W_1 \times H_1 \times D_1 \tag{9}$$

and returns:

$$o_{\text{size}} = \left( \frac{W_1 - F}{S} + 1 \right) \times \left( \frac{H_1 - F}{S} \right) \times D_1 \tag{10}$$

where $F$ is the spatial extent and $S$ is the stride of the pooling. Within the filter passing a certain stride, the maximum value within the region bounded by the filter, (which controls $F$), generating down-sampling of a given feature map. This reduces the computational time through reduction of parameters.

### 2.8 Transfer Learning

Generally in the domain of Computer Vision, the ability of a convolution neural network (CNN) is remarkably high. Despite this however, the large variety and quantities of information contained within each successive training of the weights becomes results in high computational times if training from scratch. Furthermore Computer Vision tasks such as those conducted in our study are based around a small data set which is insufficient for feature extraction through convolution. In order to address this issue, the process of transfer learning is often used [14]. Transfer learning is formally defined as follows:

Given a domain $\mathcal{D} = X, P(X)$ with feature space $X$, and a task $\mathcal{T}$ defined on $\mathcal{D}$ such that it consists of a label space $Y$ and a conditional probability distribution function $P(Y|X)$ that is learned from a given training data set consisting of pairs $x_i|y_i$, $x_i \in X$, $y_i \in Y$, the objective of a transfer is that from a source domain $\mathcal{D}_0$ along with a corresponding source task $\mathcal{T}_0$, and a target domain $\mathcal{D}_1$ with a corresponding target task $\mathcal{T}_1$, the above probability distribution $P(Y_1|X_1)$ can be learned.

Utilising this principle, within our Computer Vision task, this method relies on the pre-training of popular models such as VGG16, MobileNet and ResNet50 on a large data-set such as ImageNet [15], and applying the convolution features on our given unknown data-set with the hopes of generating reliable label sets $Y$.

### 2.9 VGG16

In the task of generating a pretrained network for our networks, the VGG16 model is of particular note due to its popularity and accuracy within Computer Vision tasks. First proposed by Simonyan in [2] the network consists of multiple convolution layers stacked with $(3 \times 3)$ filters [2] leading to a fully connected dense network structure. Firstly the input layer consists of a fixed image of size $(224 \times 224)$ with pixel information encoded to be RGB values. This input is passed through the aforementioned convolution layers where the receptive field as mentioned are restricted to a size of $(3 \times 3$, in order to capture the notion of spatial alignments such as parity (left-right etc.). Spatial Pooling is conducted after five max-pool layers spaced between the convolution layers with a $(2 \times 2)$ window. Furthermore after the convolution the information flow is inherited by three fully-connected dense layers. In [2], there are 2 layers consisting of $4096$ nodes along with the third separating the classes of the ImageNet dataset consisting of $1000$ nodes along with a final SoftMax classification layer with all layers possessing the aforementioned ReLu activation.

### 2.10 ResNet50

ResNet or Residual Networks are a means of addressing issues that arise from very deep architectures. Despite the improving ability of stacking more and more layers within networks in Computer Vision and Image recognition, there lies a saturation from which added layers no longer improve the accuracy of a constructed structure. Unsurprisingly, past such saturation points, the accuracy and performance of a model becomes increasingly poor instead of improving [16]. This is a phenomenon called "degradation" [16] for which ResNet in [16] seeks to solve. This is done through the consideration of the isomorphic nature of a shallower network to that of a deeper structure through consequent addendum of identity functions. Through such an examination, in lieu of allowing networks to learn the innate mapping approximations $\mathcal{H}(x)$ from stacked layers, residuals mappings $\mathcal{F}(x) = \mathcal{H}(x) - x$ are instead approximated through the proposed dense layer architectures. As such the original function then becomes $\mathcal{F}(x) + x$, which allows for easier learning. This residual learning process is implemented along few stacked layer consisting of a "building-block" of form:

$$y = \mathcal{F}(x, W_i) + x \tag{11}$$

where $y$ is the output of the layer stack and $x$ is the input. Then the residual function $\mathcal{F}$ is successfully learnt. In such a case this function can be graphically visualised as a "shortcut" jumping from one node to another node for which an identity is mapped. This allows for deeper structures to be considered whilst reducing the effect of accuracy degradation as previously mentioned, possibly yielding higher levels of classification performance.

In the specific case of ResNet50, the residual model is built upon the VGG16 convolution network architecture that we have previously described [2], with added layers to match retention of time complexity. To generate the residuals that we have outlined, stacks of 3 within the network is provided with a shortcut connection. This generates a $50$ layer Residual Network with identity mappings as a hope of increasing the convergence rates and classification performance of image classification compared to that of its shallower non-residual counterparts.

### 2.11 MobileNet

Given the size of very deep convolution networks like the aforementioned VGG16 and ResNet50 architectures, the computational suitability for situations requiring a lightweight but comparable classification structure becomes

increasingly difficult to develop. One such network is the MobileNet architecture proposed by [17], which adopts the use of depth-wise convolutions. This is done through application of a single filter to each input channel within the top layer of the network. Such point-wise convolutions takes a smaller window than that of the aforementioned ConvNets, with a $1 \times 1$ convolution to combine outputs of depth-wise separable convolutions. In this former case of a standard convolutions, the inputs are combined into a set of outputs for further processing be it Pooling, insertion into dense layers (etc). In lieu of such a singular process, depth-wise convolutions processes data in two layers, where a layer is designated for filtering, whereas another is required for the combination of the data outputs. In doing so, this factorisation reduces the model size [17] consequently resulting in an overall reduction of computation.

# 3 Experiment and Results

## 3.1 Model Implementations and Experimental Setup

Given the ability of the Google provided package *Tensorflow* in generating reliable architectures that interface python kernelisation with GPU processors, a series of models were constructed using the *Tensorflow* high-end API *tf.keras* module. From this both a simple ConvNet was constructed as a base-line model, along with deeper pre-trained transfer architectures with the hope of improving classification accuracy. All models were run on the Google Colab Environment taking advantage of the T4 GPU architectures. As a means of verification a similar GPU run-time was conducted on GTX1070 Nvidia GPU on an Arch-Linux python3 CUDA run-time.

### 3.1.1 Model Generations and Transfer

As a baseline model that a simple Convolution Network (ConvNet) is considered. This architecture relies on the construction of a layer stack generated from 2 convolution layers that undergoes pooling from a MaxPooling layer after dropout and batch normalisation. The network has two such stacks where the first is fed with the input dataset and the second takes the pooled ouput from the first stack as an input. This produces a 3D feature map which is subsequently flattened and fed into a dense layer of size $512$ with a final sigmoid layer for output. ADAM optimisation is the standard choice for our gradient descent with activations between layers selected to be ReLu as mentioned earlier.

We proceeded to create models incorporating deeper networks that performed well on the ImageNet Challenge such as the aforementioned ResNet, VGG16 and MobileNet. For each of these networks, an attached pooling layer and a single dense layer was subtended to the end of the transfer network. Utilising the *tf.keras* module, each of these deeper networks were downloaded with pre-trained weights trained on the *ImageNet* dataset, with freezing conducted in order to extract essential Convolution Image features. Utilising a non-aggressive rate of learning of $10^{-5}$ along with a weight decay of $10^{-6}$ Furthermore as an alternative transfer strategy, a series of fine-tuning was conducted on the lower layers of the pre-trained weights to elicit higher suitability of the transfer on our localised data-set.

## 3.2 Accuracy and Model Performance

### 3.2.1 Custom Accuracy Definition

Given the possibility that there exists multiple labels for each picture that may arise as a consequence of our classification, we define an custom accuracy measure to properly elucidate what is meant by an accurate prediction. Given a set of ground truth labels $\mathcal{Y}_t$, a prediction $\mathcal{Y}_p$ for multiple class labels is said to be correct if:

$$\mathcal{Y}_p \subseteq \mathcal{Y}_t, \quad \mathcal{Y}_p \neq \phi \tag{12}$$

Given such a definition let us limit our constructed network output to that of an element within the set of possible predictions $\mathcal{Y}_\Omega$ such that the the maximum amount of multiple labels can be predicted. In such a case we calculate accuracy $a_m$ through the percentage of such maximal elements $Y_m$ being correct over the total amount of test samples:

$$a_m = \frac{Y_m}{|\mathcal{Y}_t|}, \quad Y_m \in \mathcal{Y}_p \subset \mathcal{Y}_\Omega \tag{13}$$

### 3.2.2 Base-line model

As a base-line a simple 2 stack CNN architecture was constructed in the manner outlined above. A series of hyper-parameters were considered for more efficient tuning, spanning from the modification of the learning rates of $10^{-5}$ and $10^{-3}$, along with weight decays of $10^{-6}$ and $10^{-3}$ in different configurations. Each of these configurations were trained for 20 epochs with early stopping. This simple structure yielded surprisingly high levels of classification as seen in Table **??**.

***Table 1:*** *Training and Validation accuracies of different baseline configurations dependent on varying learning rates and weight decays*

| Learning Rate | Weight Decay | Training Accuracy | Validation Accuracy |
|---|---|---|---|
| $10^{-5}$ | $10^{-3}$ | $0.5147828$ | $0.67193425$ |
| $10^{-5}$ | $10^{-6}$ | $0.92992413$ | $0.9282618$ |
| $10^{-3}$ | $10^{-3}$ | $0.51584756$ | $0.6275333$ |
| $10^{-3}$ | $10^{-6}$ | $0.83506936$ | $0.82282656$ |

The training accuracies as a function of epoch number in each baseline configuration is presented below with Figure 3 and Figure 4 eliciting the training and validation accuracies of the differing learning rates, whereas Figure 5 and Figure 6 exhibits the differing weight decay regularisations. A comparison of these four configurations and their respective training and validation accuracies are given in Figure 7 and Figure 8, along with their training and validation losses through successive epochs in Figure 9 and Figure 10. Similarly the regularising effect of dropout at values of $0.4$ and $0.6$ was observed on generalising such a baseline model on classifying our unknown dataset.Figures 11 and 12 respectively depicts the training and validation accuracies evolving with epoch training.

#### 3.2.3 Pre-trained VGG16

Utilising the aforementioned VGG16 transfer architecture the model was trained over $15$ successive epochs, with early stopping of loss saturation. In the former case of purely weight freezing from the transfer, the accuracies over such a training period is given in Figure 13 with training and validation accuracies found to be $0.9367$ and $0.9360$ respectively. In comparison the model generated through weight re-tuning yields a differing set of training and validation accuracies of $0.9445$ and $0.9452$ respectively, seen in Figure 14 reaching lower levels of convergence.

#### 3.2.4 Pre-trained ResNet

Similar to that of the VGG16 network, the ResNet50 model pre-trained on the ImageNet dataset was transferred through weight freezing in our classification task. With the addendum dense and sigmoid layers generating a generalising effect, the model exhibited greater success in reaching high levels of accuracy $\approx 93\%$. Despite this however the rate of convergence is much more steady than that of the VGG16 networks with final training and validation accuracies of $0.942$ and $0.943$ respectively. Furthermore the re-tuning of weights conducted in the top convolution layers for better task generalisation yielded a much more sporadic convergence, but yielding higher training and validation accuracies of $0.946$ and $0.947$ respectively.

#### 3.2.5 Pre-trained MobileNet

In contrast to the deep and complex natures of the ConvNet of VGG and Resnet, MobileNet yields a remarkably smaller network structure for transfer training. Despite its smaller structure, the accuracy generated is comparable to that of both the deeper structures yielding values of training and validation accuracy of $0.940$ and $0.941$ respectively for frozen weight transfer. Figure 17 exhibits such training accuracies over the $15$ epochs of training conducted with early stopping at saturation at the $14$th epoch. Similar to the above transfer methodology from the deeper network structures, the mobilenet weights is returned for higher levels of generalisation within our given task, yielding high accuracies o $0.961$ and $0.956$ for training and validation respectively.

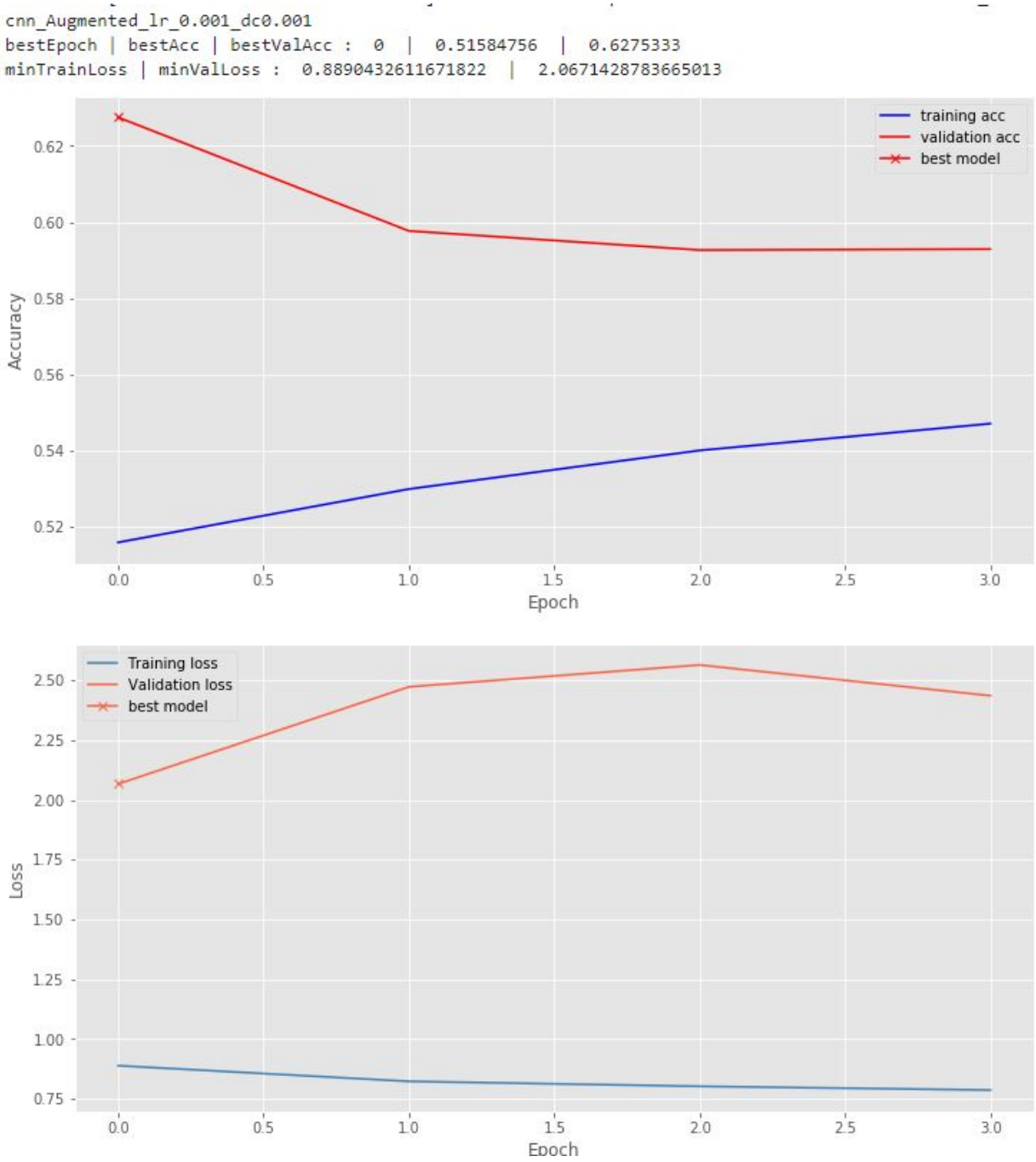


***Figure 3:*** *Training and validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of baseline ConvNet with learning rate* $10^{-3}$ *and weight decay* $10^{-3}$

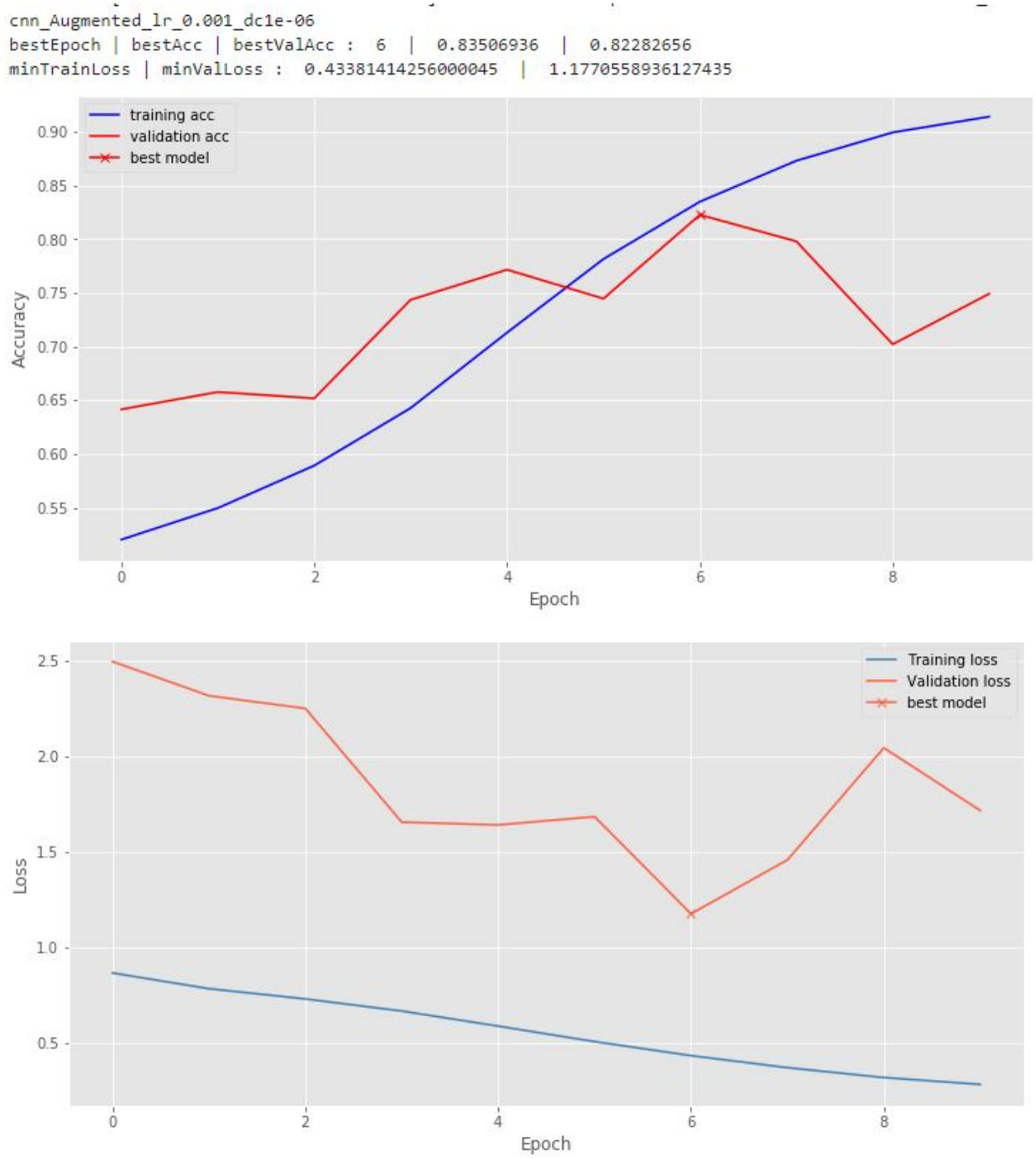


***Figure 4:*** *Training and validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of baseline ConvNet with learning rate* $10^{-3}$ *and weight decay* $10^{-6}$

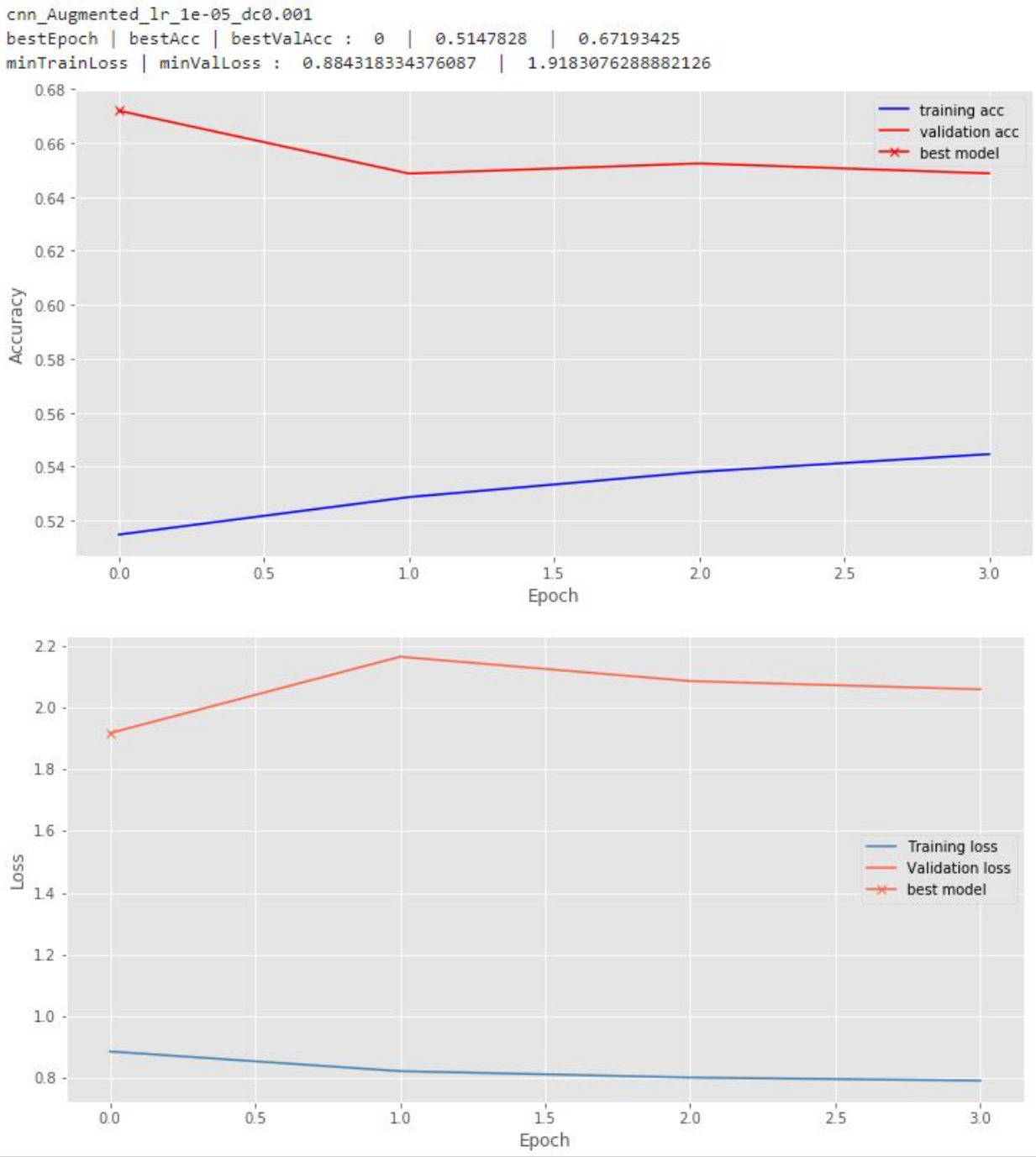


***Figure 5:*** *Training and validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of baseline ConvNet with learning rate* $10^{-5}$ *and weight decay* $10^{-3}$

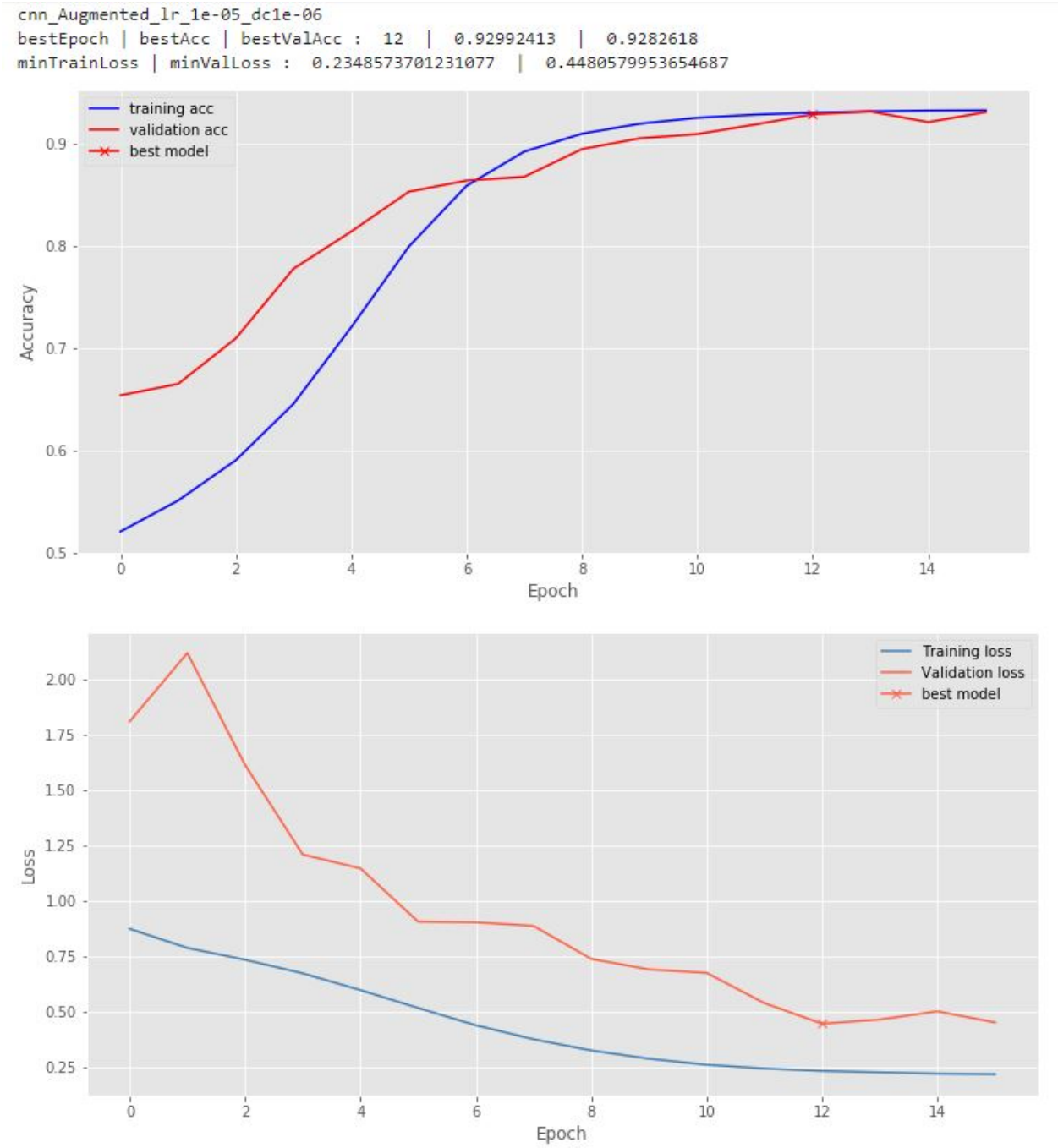


***Figure 6:*** *Training and validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of baseline ConvNet with learning rate* $10^{-5}$ *and weight decay* $10^{-6}$

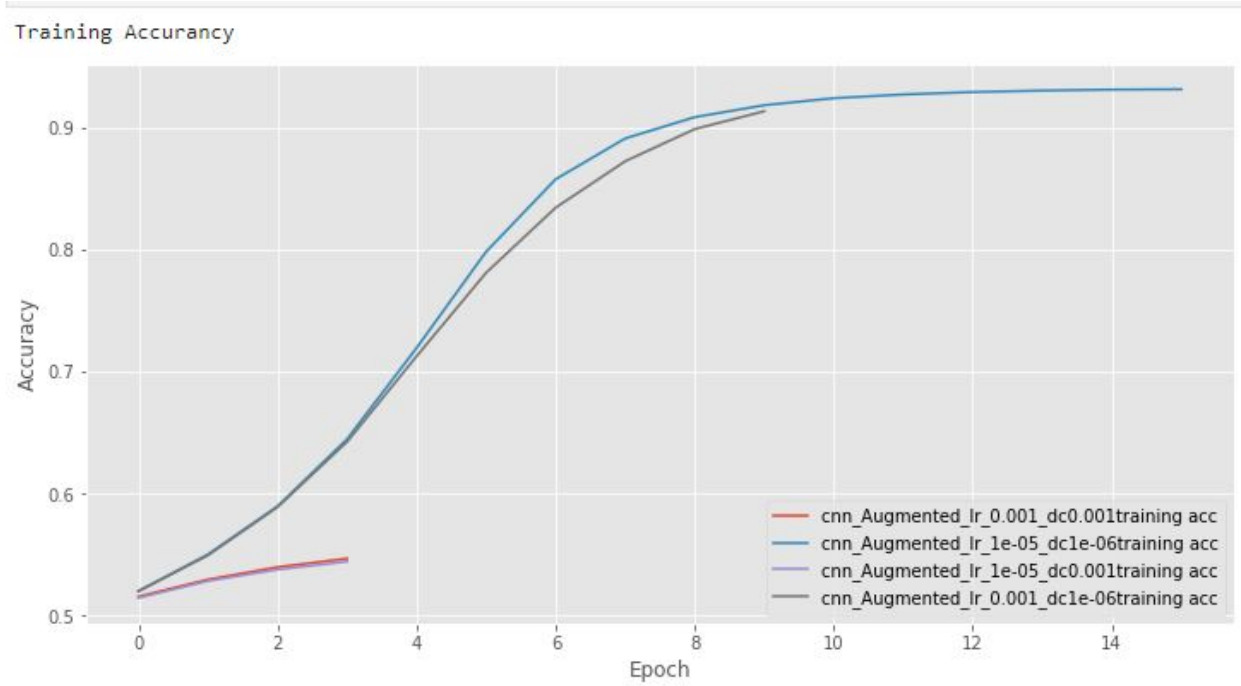


***Figure 7:*** *Comparison of training accuracy of baseline model under the differing learning rate and weight decay configurations*

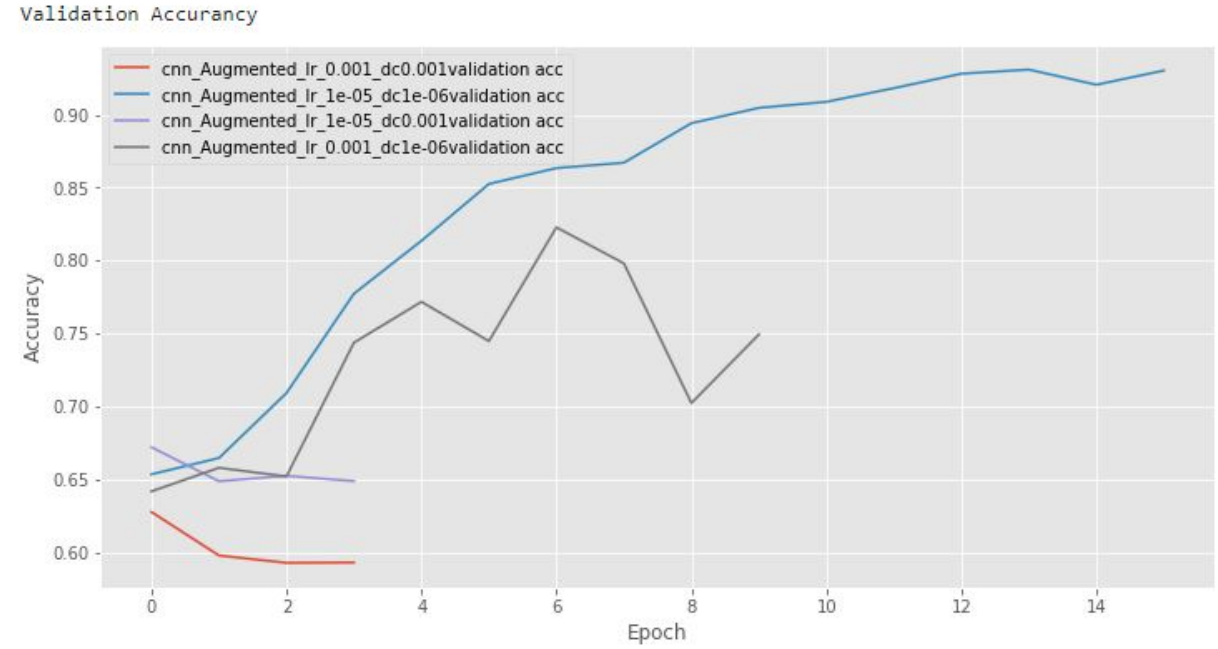


***Figure 8:*** *Comparison of validation accuracy of baseline model under the differing learning rate and weight decay configurations*

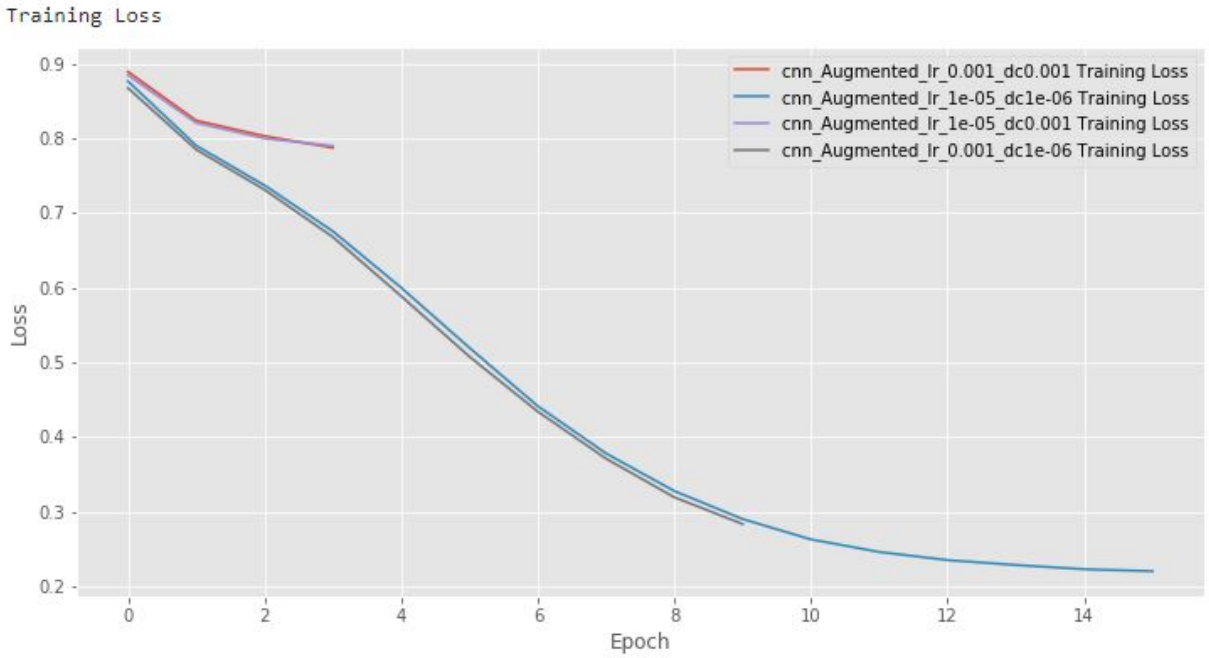


***Figure 9:*** *Comparison of training loss of baseline model under the differing learning rate and weight decay configurations*

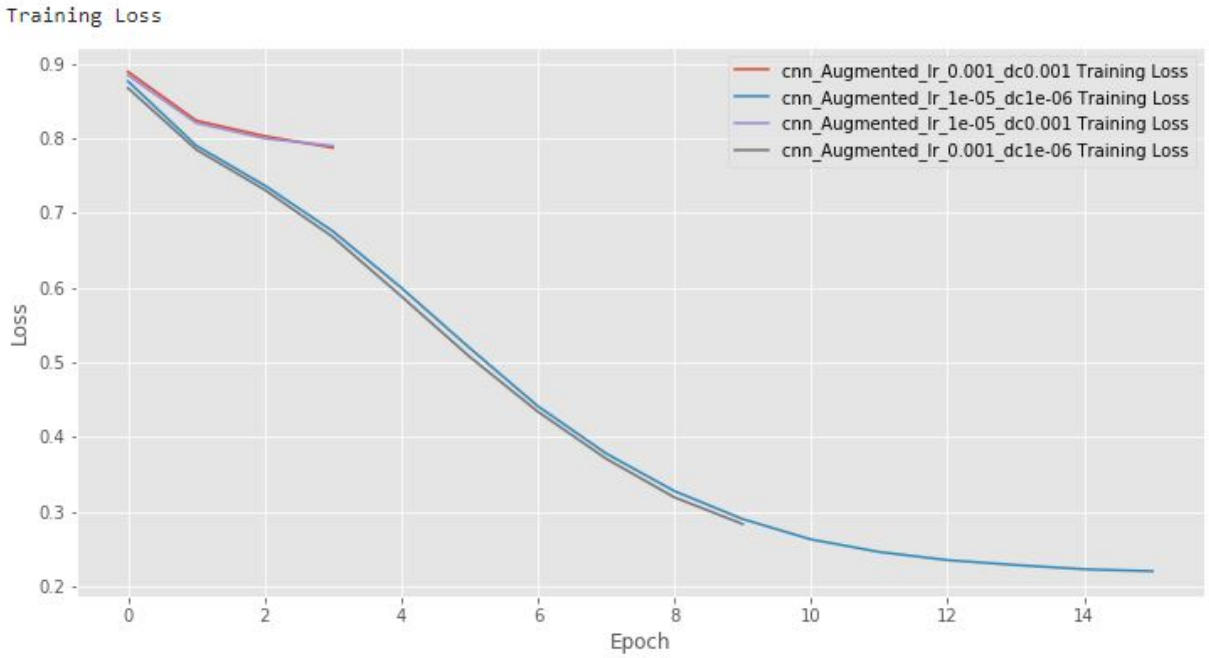


***Figure 10:*** *Comparison of validation loss of baseline model under the differing learning rate and weight decay configurations*

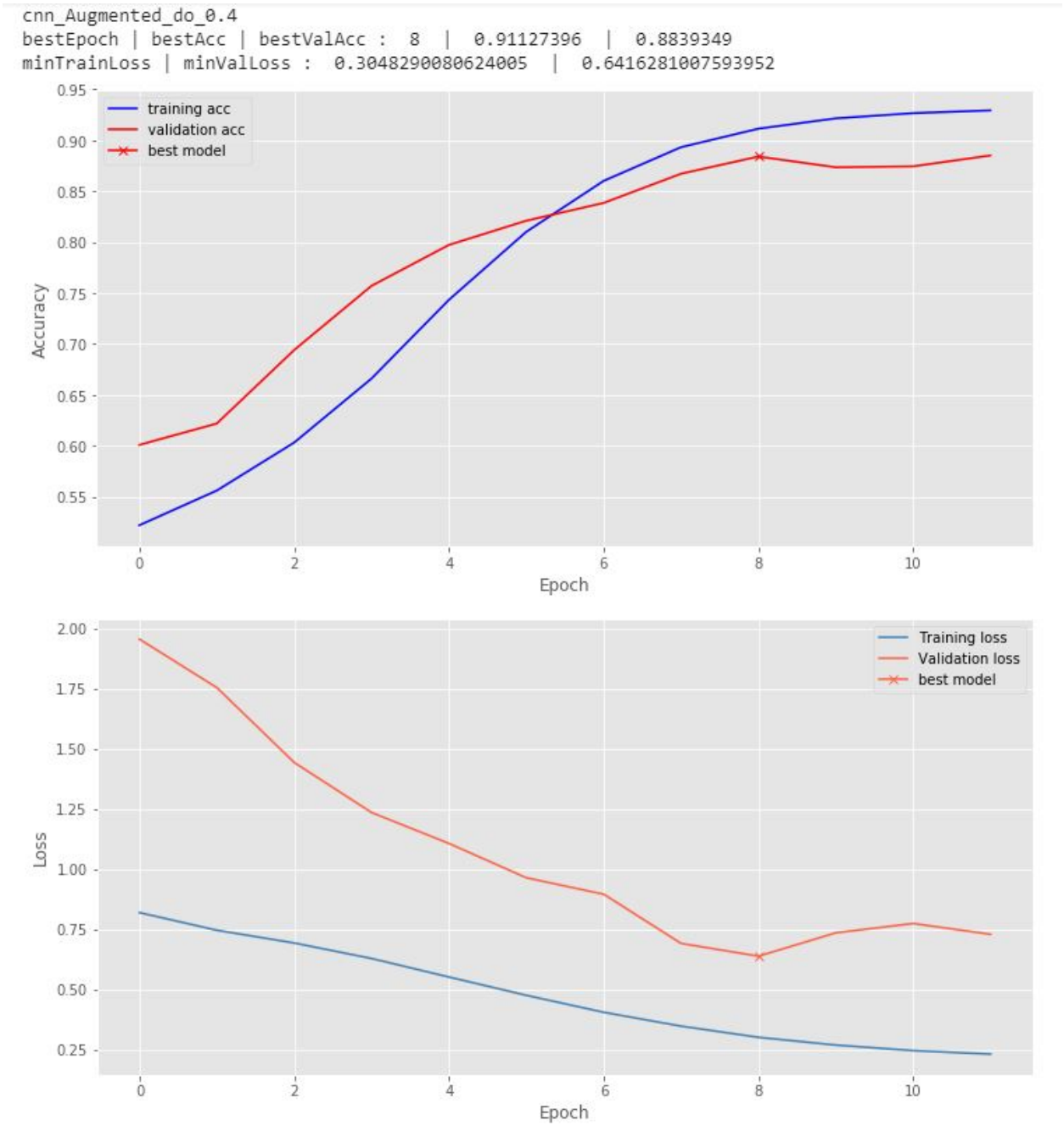


***Figure 11:*** *Training and Validation accuracies as a function of epoch number with dropout layers tuned to a rate of* 0.4

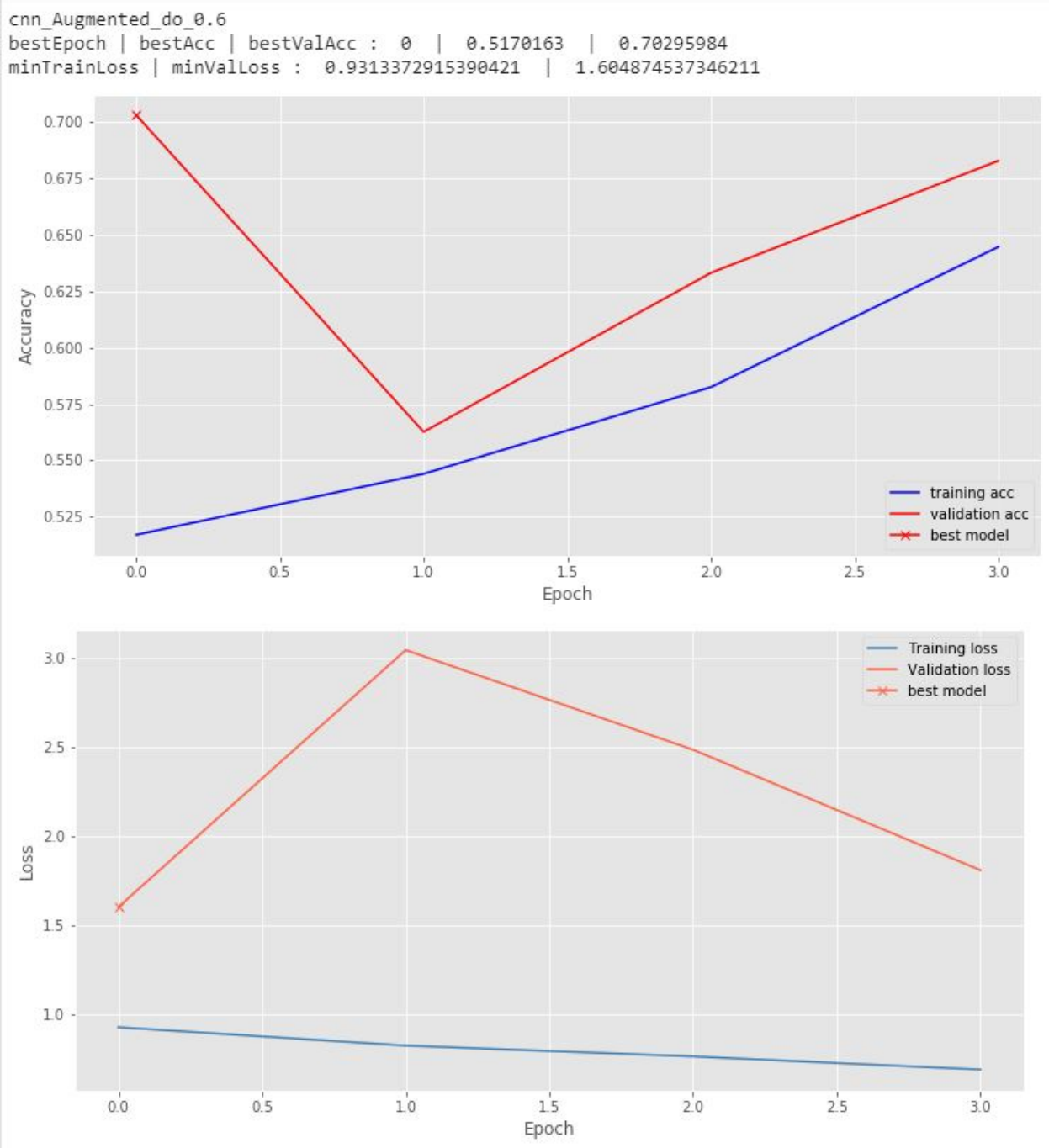


***Figure 12:*** *Training and Validation accuracies as a function of epoch number with dropout layers tuned to a rate of* 0.4

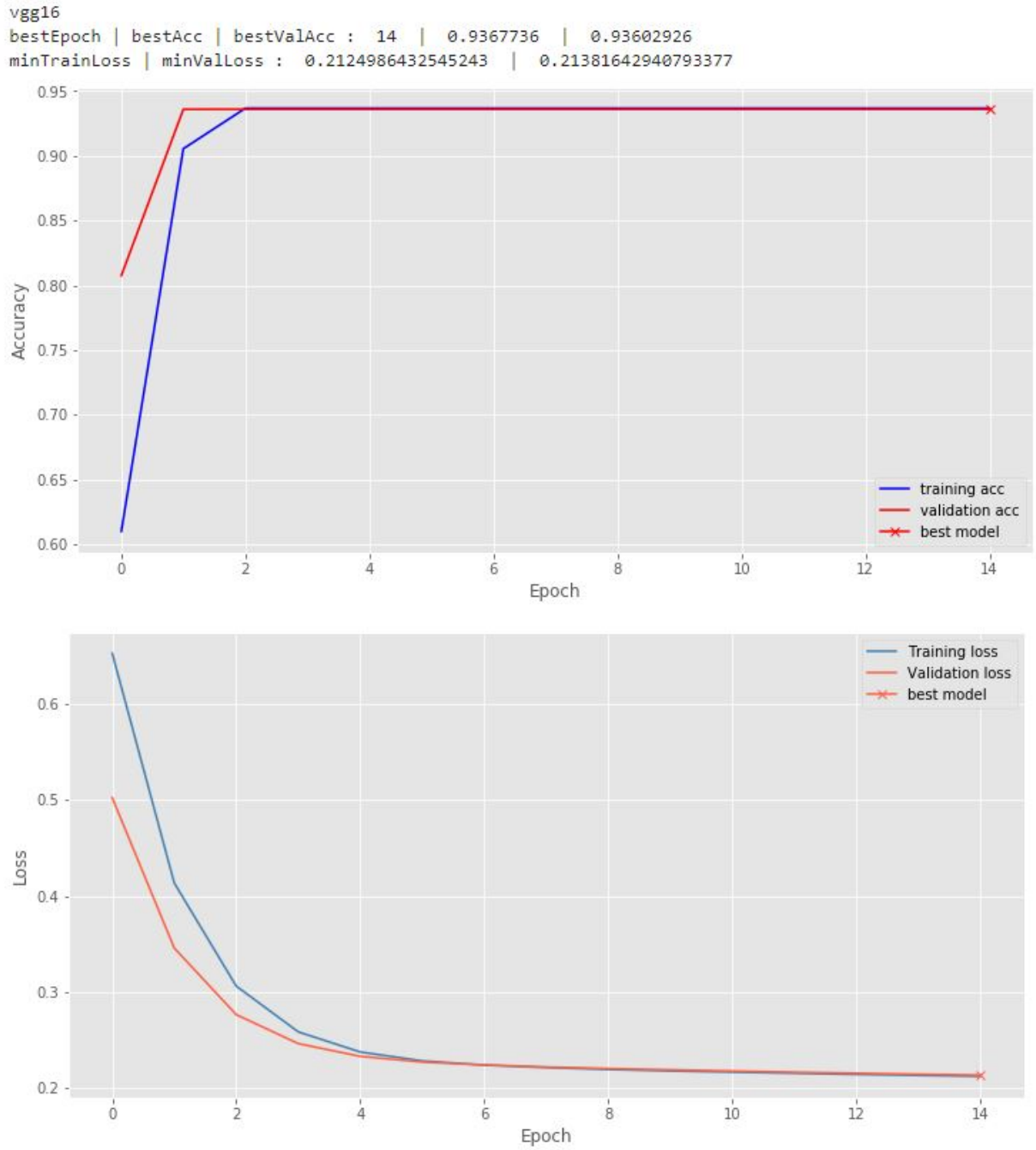


***Figure 13:*** *Training and Validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of transferred VGG16 deep learning architecture through top layer freezing.*

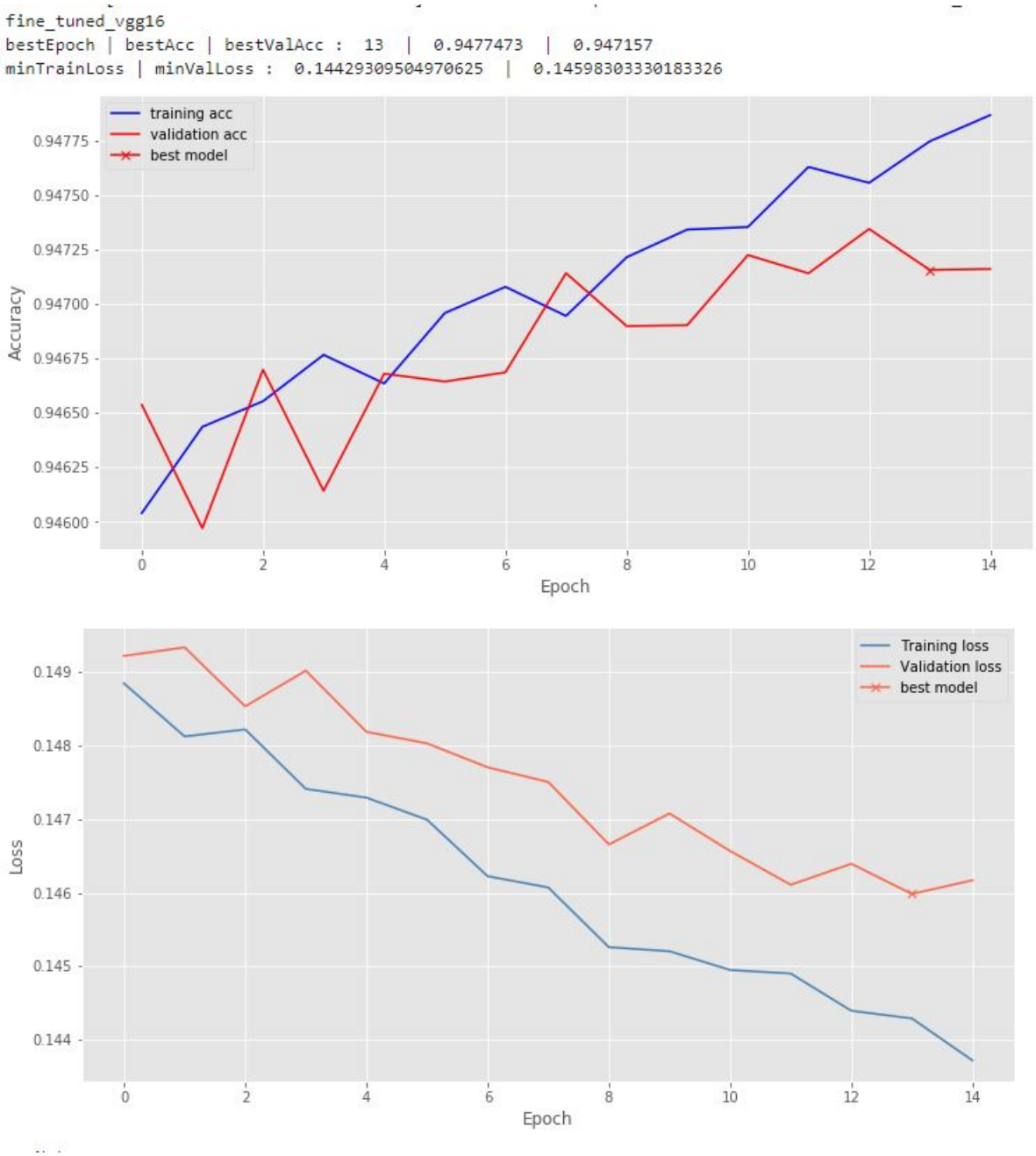


***Figure 14:*** *Training and Validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of transferred VGG16 deep learning architecture through top layer freezing along with weight retuning.*

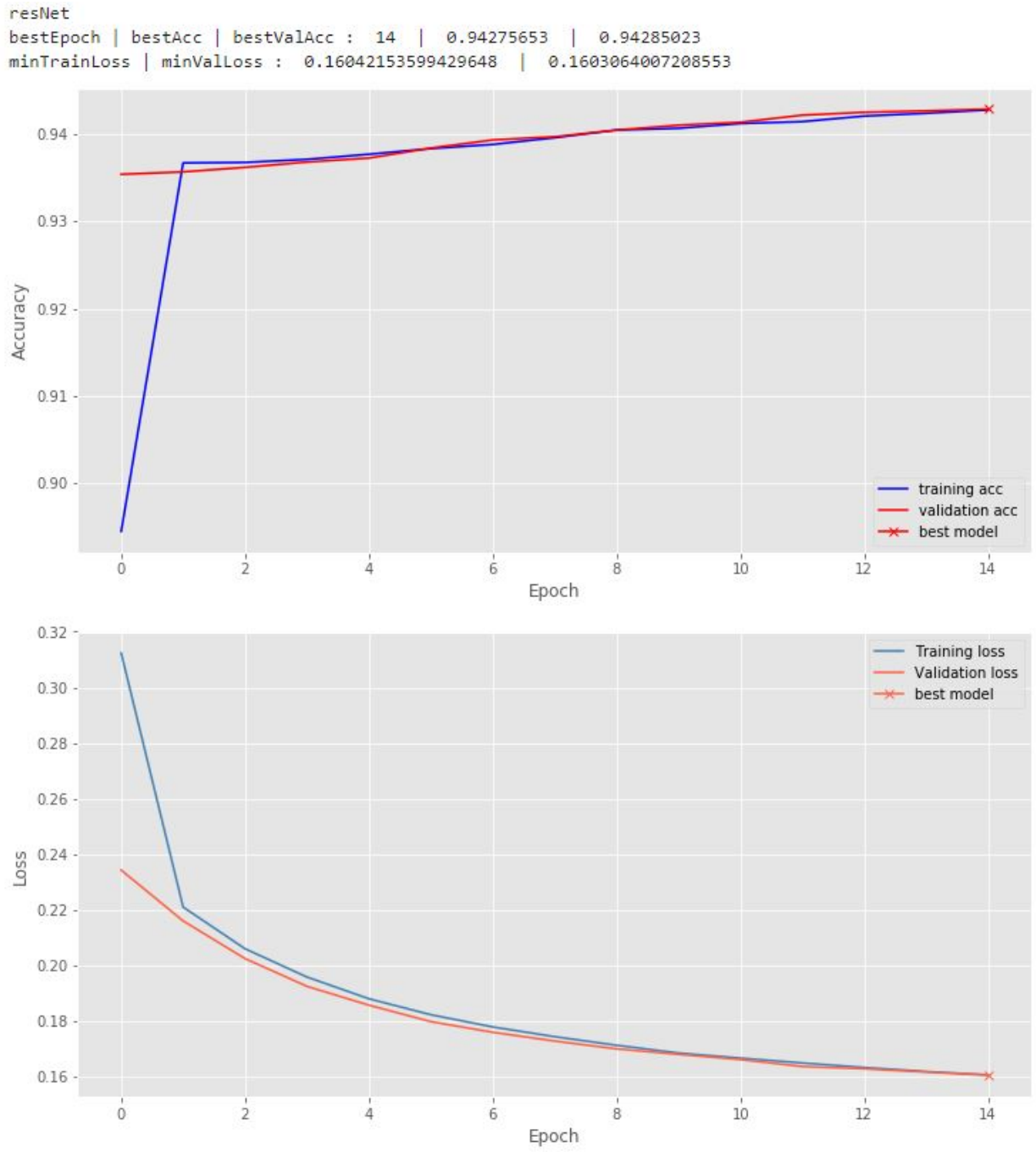


***Figure 15:*** *Training and Validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of transferred ResNet50 deep learning architecture through top layer freezing.*

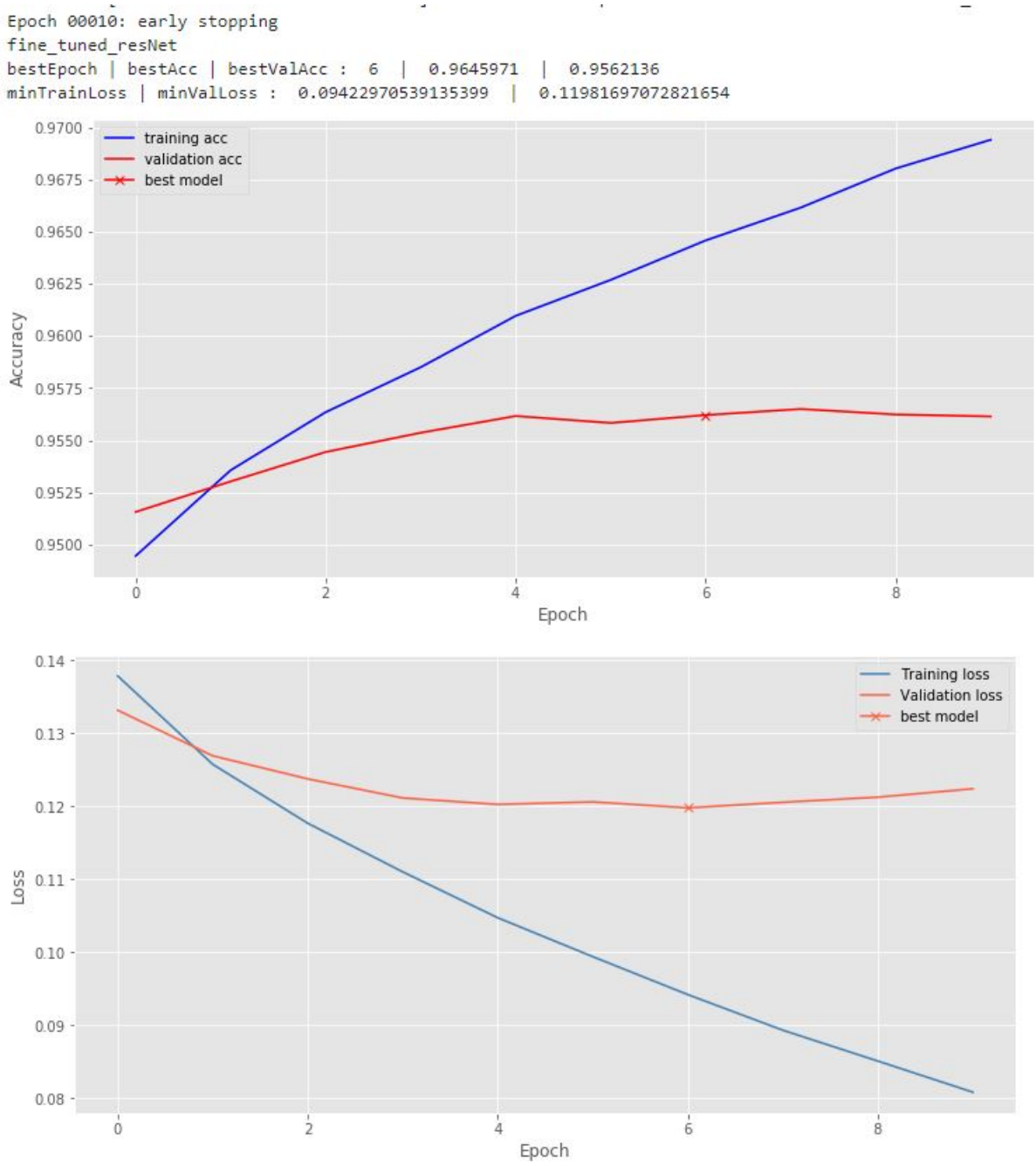


***Figure 16:*** *Training and Validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of transferred ResNet50 deep learning architecture through top layer freezing along with weight retuning.*

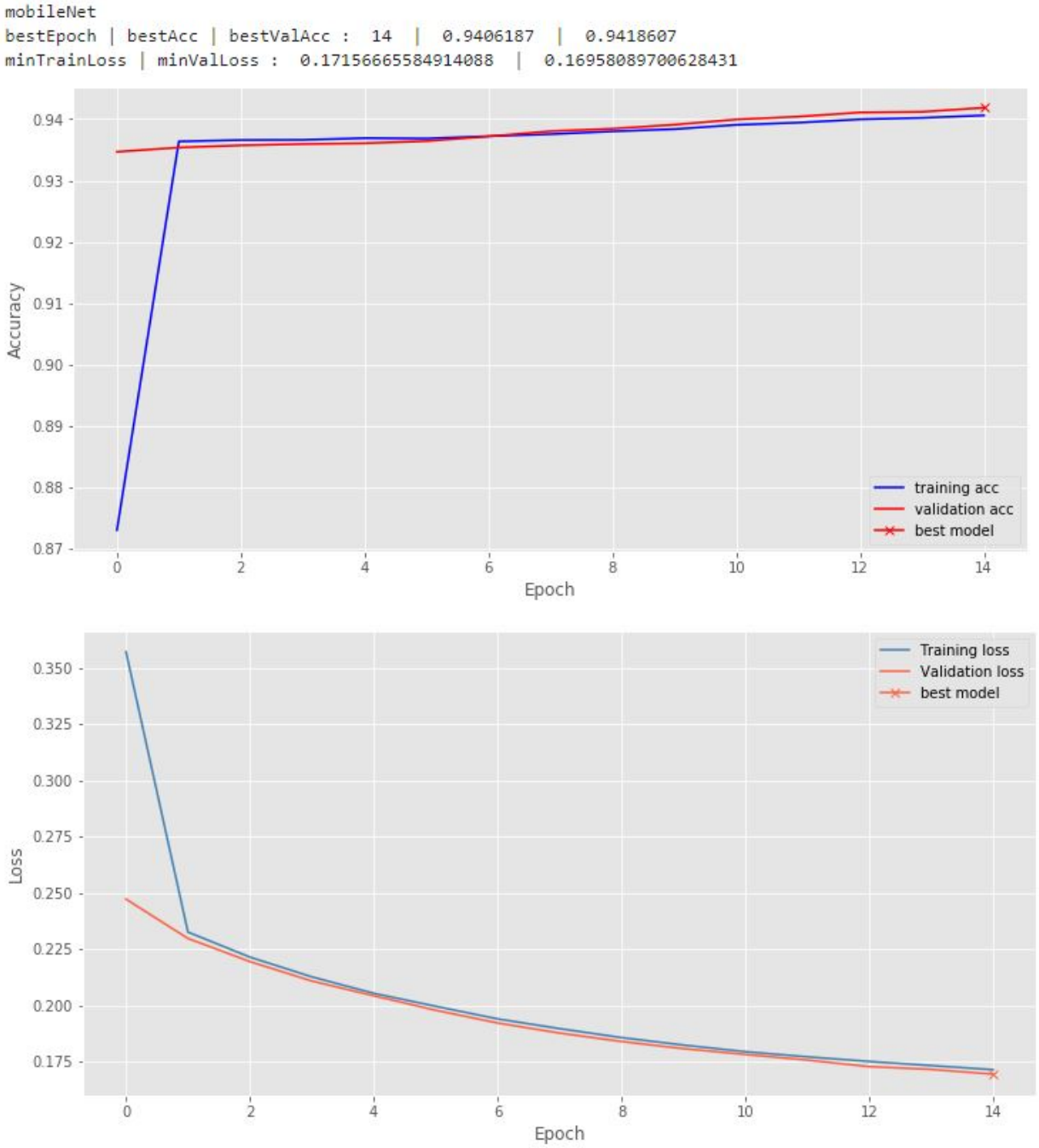


***Figure 17:*** *Training and Validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of transferred MobileNet deep learning architecture through top layer freezing.*

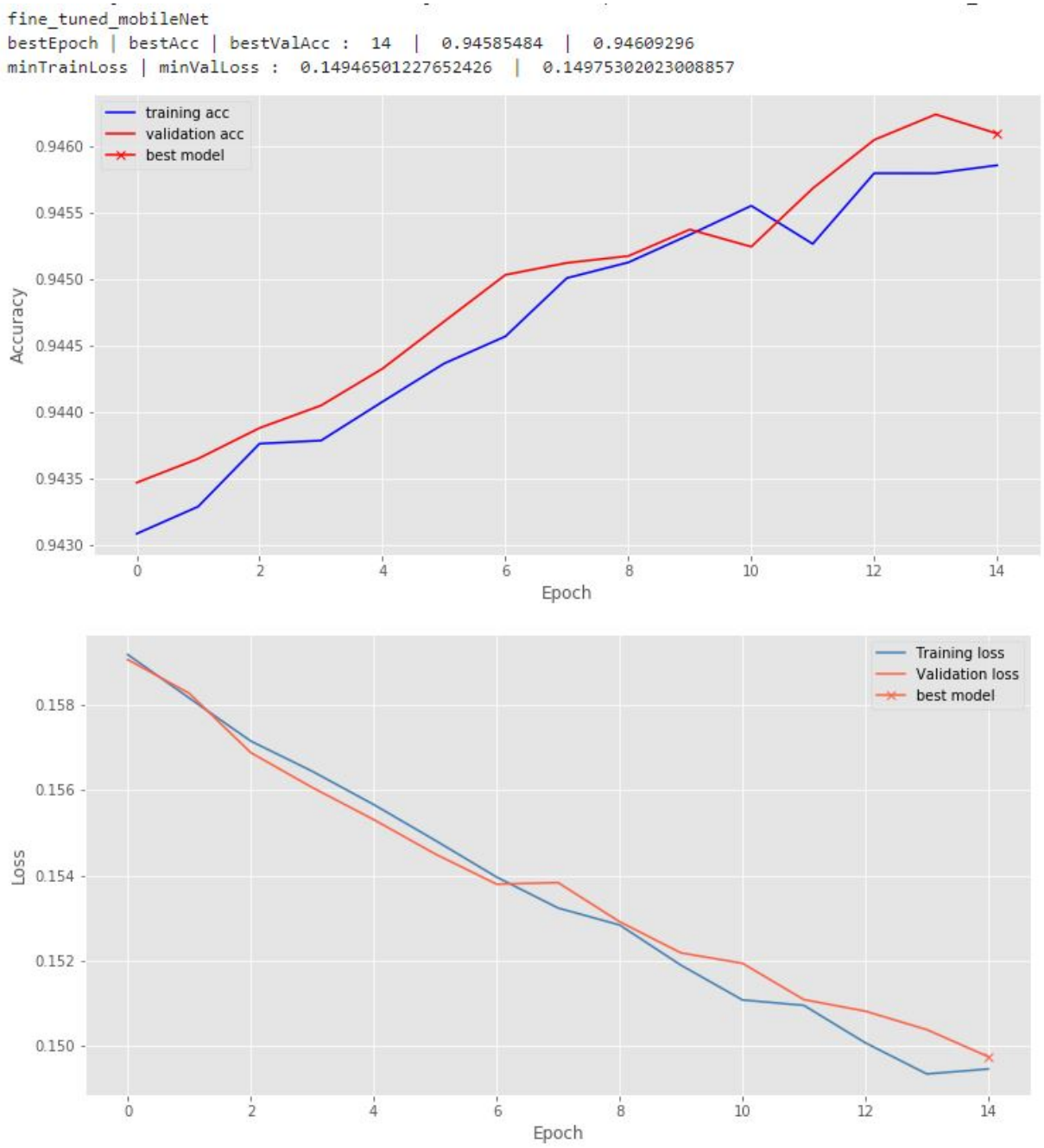


***Figure 18:*** *Training and Validation accuracy as a function of epoch (TOP) along with loss as a function of epoch (BOTTOM) of transferred MobileNet deep learning architecture through top layer freezing along with weight re-tuning.*

## 4 Analysis and Discussion

In order to evaluate the ability of our constructed networks in generating accurate models in relation to our custom definition of accuracy, a variety of observations can be drawn from our given results by examining the above figures. This is clearly evident in the creation of different configurations of a basic base-line ConvNet architecture that was designed. As exhibited in Figure 3 and 4, a heightened weight decay hyper-parameter yields slower convergence and classification accuracy however possibly yielding higher levels of generalisation within our dataset. This effect is well remarked in literature [18] and its unsurprising effect promises higher generalisation towards unknown dataset within the future. Comparatively, a higher learning rate yields a faster rate of convergence with the accuracy gradient

exhibiting a sharp incline at a more aggressive learning rates. This is evident in a comparison of Figure 3 and Figure 5. Once again this effect is well known within literature [19] and serves as a benchmark on the constructed ConvNet in reproducing well known effects. Despite this relative success in producing high accuracies for training, the validation accuracy remains rather low. We suggest that this is due to the sparsity of the dataset in comparison with larger datasets such as that of ImageNet [15]. Furthermore comparable literature of [16] suggests higher model performance at deeper networks. Such deeper networks, in particular those of transfer learning models of VGG16, ResNet50 and MobileNet yields higher but comparable results with faster rates of epoch convergence at the sacrifice (in the case of VGG16 and ResNet50) computational time. This can be clearly seen in the sharp convergence rates of all three transfer structures yielding a rapid jump past the first two epochs of training. Despite the efficacy of all three networks and their similarities in generating high performing classifications, a variety of differences can be observed due to their different constructions. This is most noticeable in both the rapidness of convergence of loss and accuracies, with VGG16 exhibiting higher levels of saturation at the cost of computational time in comparison to that of ResNet, which belies a slow yet steady classification accuracy convergence at finer increments. We suggest that this is due to the heightened levels of layers with added complexity, with the shortcut connections leading to more refined computations of the accuracy. This is also reflected within the mobileNet construction, which remarkably yields a similar performance to that of the high performing ResNet structures commonly described within literature [16]. Despite the relaltive success of such deep architectures in transferring top level weights, the fine-tune adjustment to increase generalisation to our given dataset yields overall heightened classification accuracies both in terms of training and validation. In doing so however, there are a wide variety of effects that can be observed within such fine tuning. One such effect of note is the sporadic nature of accuracy through successive epoch training, which is especially noticeable in that of the fine-tuned ResNet, yielding accuracy degradation after a spike. We cannot explain this phenomenon at first notice and further examination into such effects is recommended for future study. Furthermore, a multitude of improvements can be conducted, with consideration of adding multiple layers along with $k$-fold cross validation as an addendum to our given transfer networks, which may improve overall performance and decrease overestimation of accuracy from a singular validation procedure that we have conducted. Also, due to the custom defined accuracy metric that is required, a series of ambiguity arises in consideration of labels that are not maximal spanning. Our sigmoid prediction from our final dense layers suggest a multiple label with most probable labels for classification which may not be the case. An example of this is the presence of singular labels within the classification, which may be mislabelled to possess more than one label. Furthermore, the computational expense of running multiple models is remarkably high, and no grid search procedure was conducted to examine for hyper parameter tuning. This is especially true for that of the transfer networks, whose "out-of-the-box" architecture yields the inability to fine-tune parameters as we would like. In recommendation, future studies would disseminate and modify such transfers to yield possible finer level considerations of features extracted from the top layer constructions of the given transfer architectures.

## 5 Conclusion

It can be seen that the ability of Deep Learning architectures to create classifications is varied depending on the computer vision task that is required. In our case of generating a label set for a series of images, the constructed learning structures is dependent on what sort of architecture is designed. Architectures such as that of transfer structures provide a transitive consideration of the power of the method of transfer in examining images, which may generally be difficult to do due to the limited quantities of samples within provided dataset. Although such structures are seen to be more efficient than that of a shallower base-line ConvNet and their configurations; there are a variety of differences in the rate of convergences and computational times. It can be seen that even though the deeper networks such as ResNet warrant consideration as a canonical method of Computer Vision tasks, it may be more beneficial to utilise a network with lower complexity such as that of MobileNet. Through such an exploration into the differing methods along with a variety of aforementioned improvements, we are able to suggest and design higher performing Network structures for future network designs as a means of Image classifcation.

## References


[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90:119–133, 2019.

[4] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[5] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[6] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[7] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[10] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

[11] Nitish Srivastava. Improving neural networks with dropout. *University of Toronto*, 182:566, 2013.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[13] Naila Murray and Florent Perronnin. Generalized max pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2473–2480, 2014.

[14] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2208–2217. JMLR. org, 2017.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[18] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[19] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.