Building Multi-Layer Perceptrons (MLPs),Convolutional Neural Networks (CNNs) with Tensorflow/Keras

Intro to Deep Learning Assignment 1

Yutao Liu (s4213211)
Nael Belhaj Haddou (s4431278)

1 Task 1: Learn the basics of Keras API for TensorFlow

1.1 Instructions

1. This experiment requires us to get familiar with Tensorflow and Keras deep learning library and use it to build neural network models for classification and regression tasks on image datasets(in this task we use MNIST and CIFAR - 10 as examples). By comparing various MLP and CNN architectures, we could gain parctical experience in hyperparameter tuning and model experimentation to optimize the performance of model, applying those knowledge to develop more complex CNN for further task.

1.2 Experiment

1. We take the same steps to create a CNN model as the guide book told, note that the Input Shape used by MNIST and CIFAR-10 is different. After the initial training, we take measures to improve the models such as increasing the number of network layers and neurons for the MLP model, increasing the number of convolutional layers and the number of filters per layer for the CNN model to observe their effects on accuracy and loss.

2. Separate the MLP and CNN model creation and training codes in the first step to prepare for the hyperparameter tuning and model experiments. We train the MLP model with the MNIST dataset first, store different hyperparameter combinations (including activation function, optimizer, learning rate, and Dropout ratio) in array, and save the results in Dataframe form after training and output the 3 sets of hyperparameter combinations with the best accuracy. Continuously adjust the hyperparameters to determine the best settings to provide a reference for subsequent training of the CNN model.

3. Put the three best hyperparameter combinations on the MNIST dataset into the CIFAR-10 dataset for training and testing, and observe and record the performance changes under different datasets as well as the accuracy, loss, overfitting and training time differences, try to figure out the reasons for different performances. At the same time, we will also test the same model on two datasets at the same time to observe its performance on different datasets.

1.3 Observations

1. Experiments on Network Architecture:

(a) Initial Version:

- MLP: Three neural layers with 128, 64, and 10 neurons in each layer.
- CNN: Two convolutional layers with 32 and 64 filters in each layer.

(b) Intermediate Version:

- MLP: Three neural layers with 256, 128, and 10 neurons in each layer.
- CNN: Two convolutional blocks, with two convolutional layers in each block, with 32, 32, 64, 64 filters in each layer.

(a) Initial Version

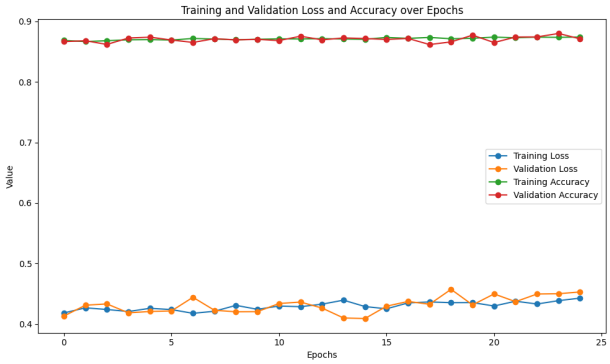
(b) Intermediate Version

(c) Advanced Version:

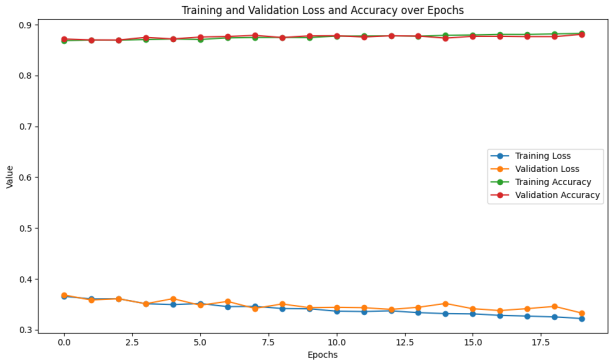
- MLP: Four neural layers with 256, 128, 64, and 10 neurons in each layer.
- CNN: Three convolutional blocks, with three convolutional layers in each block, with 32, 32, 64, 64, 128, 128 filters in each layer.

(d) Advanced Version(SGD):

- Same setting with Advanced Version but the optimizer of the model was replaced by SGD (learning rate=0.01, momentum=0.9) from RMSprop.



(a) Advanced Version



(b) Advanced Version(SGD)

2. Experiments on Different Hyperparameters:

   (a) Hyperparameter Tuning in the MNIST Dataset:

   - Set six sets of hyperparameters for training at a time, select the three sets with the best training effect, and continue to adjust the hyperparameters for the next training on this basis until the training effect starts to deteriorate, then count and select the three sets of hyperparameters with the best effect.

Activation Function	Optimizer	Learning Rate	Dropout	Val Accuracy
ReLU	Adam	0.0005	0.2	88.62%
ReLU	RMSprop	0.0005	0.2	88.35%
ReLU	RMSprop	0.0005	0.1	87.80%
ReLU	RMSprop	0.001	0.3	87.75%
Tanh	RMSprop	0.001	0.3	87.68%
ReLU	Adam	0.001	0.3	87.40%
ReLU	RMSprop	0.00005	0.05	86.78%
ReLU	RMSprop	0.00005	0.15	86.33%
ReLU	Adam	0.00005	0.15	86.00%

   (b) Hyperparameter Tuning in the CIFAR-10 Dataset:

   - Use the three sets of hyperparameters with the best training effect of the MNIST dataset to train the CIFAR-10 dataset and compare the differences in training effect. Then, find the best hyperparameter combination for training the CNN model in the same way. (Since the initial code training effect was not good, continue testing on the basis of deepening the network structure of the CNN model).

Model	Optimizer	Learning Rate	Dropout	Val Accuracy
Improved	SGD	0.005	0.3	78.54%
Improved	SGD	0.005	0.1	78.21%
Improved	SGD	0.007	0.1	76.90%
Improved	SGD	0.005	0.2	74.19%
Initial	Adam	0.01	0.2	73.94%
Improved	SGD	0.007	0.2	73.94%
Initial	Adam	0.00005	0.15	73.74%
Initial	RMSprop	0.00005	0.15	73.52%
Improved	SGD	0.007	0.2	73.51%

3. Experiments on Same Models, Different Datasets:

   - Test the same model on Fashion MNIST and CIFAR-10 to see how they perform on different datasets and output the results as a line graph.

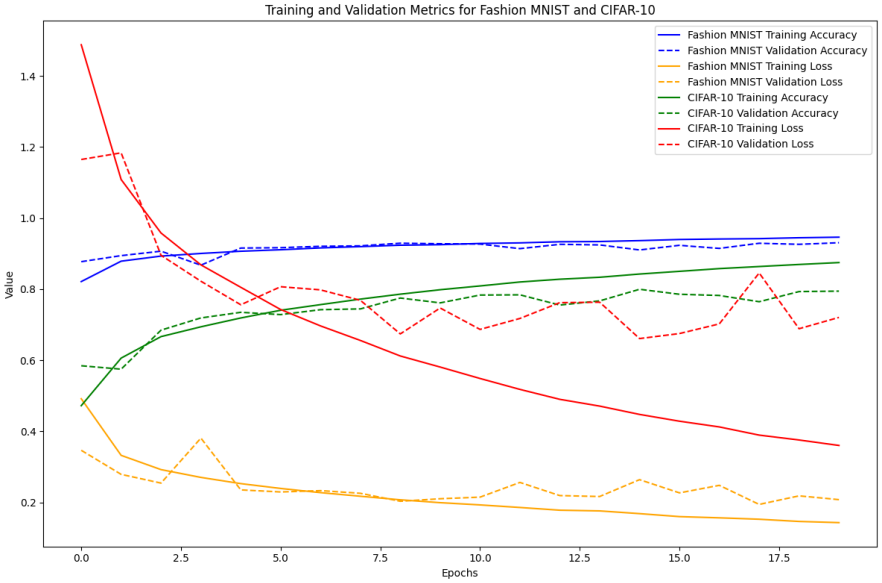


Figure 3: Training Performance on Different Datasets

1.4 Conclusion

1. The influence of network structure complexity on training effect

   (a) In the initial version, the structures of both models were relatively simple, making it difficult to capture the complex features of the data set and further model them. Therefore, their performance was poor and under-fitting. In the intermediate version, we improved on the MLP structure. The number of neurons is increased, convolution blocks are added to the CNN structure, and the number of filters is also increased. This setting can better capture the complex features of the image, so the loss is significantly reduced and the accuracy is improved, but the model Stability and generalization capabilities are not yet optimal.

   (b) In the Advance version, we increase the deeper network structure allows the model to learn richer features, the loss continues to decrease and the accuracy increases. However, the more adaptable learning rate adjustment mechanism of the RMSprop optimizer may lead to slower convergence and less obvious loss reduction. In the end, we changed the optimizer from RMSprop to SGD with momentum, which avoided the local optimal stagnation problem of RMSprop and found the lower loss point faster. Finally, we obtained the best convergence effect with both loss and accuracy. The curve shows that the model has reached a moderate fitting state and has good generalization ability, with almost no signs of overfitting.

2. The influence of hyperparameter combination on training effect

   (a) Effect on MNIST dataset:(1) Higher learning rate may cause the model to skip the optimal point during training and convergence is unstable; Lower learning rate may cause the model to converge slowly or even fall into a local optimum; (2) Adam and RMSprop usually perform better on MNIST because they have the ability to dynamically adjust the learning rate, while SGD may perform poorly when the learning rate is not good; (3) The MNIST

model is not suitable for a high Dropout rate because it may cause too much information to be lost when learning data features. (4)For a simple dataset such as MNIST, the non-linear performance of ReLU is sufficient to extract the main feature information.

(b) Effect on CIFAR-10 dataset:The optimal hyperparameter combination of the MNIST dataset performs poorly on CIFAR-10. The reasons may be: (1) CIFAR-10 images contain rich colors, textures, and details, and the model requires a moderate Dropout rate to prevent overfitting; (2) The model on CIFAR-10 requires a more stable optimizer (such as SGD) to cope with higher complexity; (3) CIFAR-10 requires a relatively higher learning rate to prevent slow convergence.(The Initial version of CNN model was used when testing the best hyperparameter combination for CIFAR-10, so the accuracy is poor, but it can still count features).ecture about the appropriate hyperparameters for the two datasets. To adapt to CIFAR-10, models usually require deeper structures, higher regularization, and appropriate optimizers and learning rates to cope with more complex feature extraction and higher risk of overfitting.

2 Task 2: Develop a "Tell-the-time" network

2.1 Instructions

1. This task had us work on a "tell-the-time" network. A network that would be fed a picture of a clock that will generally be angled similarly but with any rotation applied. We were tasked with experimenting with various types of networks, architectures and hyperparameters. Once we had tested multiple networks we were to find a network that would minimize common sense error, the absolute time difference between our prediction and the true label.

2.2 Experiment

1. As suggested, we divided our datasets in 80% training data, 10% validation data and 10% test data for the first part. For the second part, we did as suggested again and trained using an 80% training and 20% test split. The test data was only used once per model to compare them (no hyperparameter tuning was done using it).

2. All model architectures are using the same convolution "pattern", their convolution layers are always organized as two convolution layers with ReLU with twice as many filters as the previous pattern (starting at 32 filters on the first pattern). The kernel sizes are 3 by 3. This is followed by a 2 by 2 max pooling layer. This pattern was then repeated multiple times (twice for most architectures).

3. All models were trained using stochastic gradient descent and with a learning rate of 0.01 for 150 epochs unless specified otherwise. Additionally any classifier output used sparse categorical cross-entropy loss and regressor outputs used mean squared error loss. Lastly, unless otherwise specified the batch sizes were 32.

4. In the first part of the task we trained 9 classifier models. We made 4 parameters vary between models: the number of convolution patterns, the number of hidden layers, the learning rate and

the batch size. The values we tested for learning rate and batch size were 0.1 and 256, 0.05 and 128 along with 0.01 and 64 respectively. We used these hyperparameters as a pair because we believe that the increase in batch size should help stabilize the gradient descent. We experimented with architectures of 2 convolution patterns and 2 hidden layers, 2 convolution patterns and 3 hidden layers and 3 convolution patterns and 3 hidden layers. As the output had 720 classes, all hidden layers had 512 neurons and had a dropout layers with dropout rate of 0.25 interspersed.

5. We then trained 3 regressor models, we used a different convolution pattern thrice. We added a dropout layer with 0.25 dropout rate after each max pooling, we also changed the stride, the first model had stride 1 to have a regression baseline, the second had stride 2 by 2 and the last had stride 2 by 1 and 1 by 2 kernels one after the other. After that we had 3 hidden layers with 256 neurons each. To prevent overfitting we put a 0.25 dropout layer between each hidden fully connected layer.

6. We experimented on 4 two headed models, attempting all combinations of regression and classification with the same architecture. The architecture used 2 convolution patterns with dropout, followed by 3 hidden layers of 256 neurons with 0.25 dropouts between each layers. We used a similar architecture in most models as it seemed to perform well on the smaller dataset.

7. We tested 2 architectures with a label transformation. The first architecture predicted the sine and cosine of each clock hand. The second architecture only used the hour hand's sine and cosine to compute the complete time. The models used 2 convolution patterns with dropout, leading to a single dense layer with 256 neurons for the two head model and 2 dense layers for the four head model. Then the network branched out for 2 more hidden layers (each half the width of the one before) with dropouts interspersed per head.

8. Finally, We tested 5 variations of our best performing architectures after training them for 300 epochs. The first three models are variation of our two headed label transformation model and the last two are variations of our two headed classification model. The first variation is simply adding one more hidden layer per head, attempting to tap into any underfitting. In the second variant we doubled the convolution kernel size from 3 by 3 to 6 by 6 in hopes that this larger kernel somewhat compensates for the larger images. The third variant tried to achieve the same thing but did so by adding a 2 by 2 average pooling layer before convolution. The fourth variant was based on the two headed classification model and added a layer per head, also in an attempt to fit the data better. Lastly the fifth model was also based on the two headed classification model and added a convolution pattern.

2.3 Observations

1. It is important to note that loss graphs of all the different models cannot be compared 1 to 1. The different numbers of heads and differences between regression and classification can lead to very different values for models that have similar common sense error. Thus, when comparing networks of different types we will mainly use common sense error as that is the value we were trying to optimize in this task.

2. Most of our tests were done on the 75 by 75 dataset, inevitably we suffered quite a massive drop in performances when applying our architectures to the larger resolution dataset. We believe this

is caused by the increase in input data complexity while the model is kept the same. This leads to a model that is underfitting.

3. The classifier model is the one we did the most experiments on in hopes of getting its common sense error under 1 hour. Although we could not achieve that, we still gathered many interesting insights.

   (a) The first thing that surprised us after our experiments with the classifier models was the overall very poor performance it offered. All validation accuracies we had were under 10% and our common sense error was over 2 hours for all classification models with the best model having a common sense error of about 2 hours and 14 minutes. Even across all our tests on the simpler dataset, the classifier common sense error never went far below 2 hours. The model that performed best in terms of common sense error is the simplest, the architecture with 2 convolution patterns, 2 hidden layers, a learning rate of 0.1 and a batch size of 256. However the difference in mean common sense error between the models we tested was less than a minute, implying that all these models are approximately equal (though at equal accuracy a simpler and cheaper model is superior).

   (b) We had two hypothesis on the cause behind the plateau in performance of this architecture, our first thought was that it was simply that the model was overfitting the data as the training accuracy and loss were very good on later epochs but considering the simplicity of our model and the somewhat better performance and lower overfitting of more complex models (as can be seen in figure 4) we don't currently believe this to be the cause. The second idea we considered was that this plateau possibly originates from the use of sparse categorical crossentropy as the loss function while we're trying to optimize for common sense error. Their relation is somewhat tenuous as crossentropy doesn't account for how close the model was to the right answer while it is an extremely important factor for common sense error.

   (c) We wished to test different learning rates with different model complexity because we believed higher learning rates may help more complex models escape local minima. Looking at our results it does not seem that is the case, in fact the model that converged the best had one of the highest learning rates. It even appears that smaller learning rates could help more complex models better. As can be seen in figure 4, when comparing each of the levels of complexity with learning rates 0.01 and 0.05 it appears that the more complex model, converged faster with a lower learning rate, contrarily to the simpler models.

4. The regression models were used to test the effects of stride. We had quite a hard time finding any architecture of this type that fit the 150 by 150 dataset. On the smaller dataset they were quite effective but here they almost all converged to a common sense error of 2 hours and 14 minutes like the classifiers. We believe it's because the complexity of the problem compared to the 75 by 75 images has increased beyond what these models can handle. However we found a model that converged to a lower error of about 1 hour and 29 minutes. That model used strides of 2 by 1 and 1 by 2 in its convolution patterns. We are unsure of the reason this architecture performed better because the 2 by 2 and 1 by 1 strides performed very poorly. All these models converge very fast as shown in figure 5.

5. Because of the poor performance of the classification and regression models, we had little

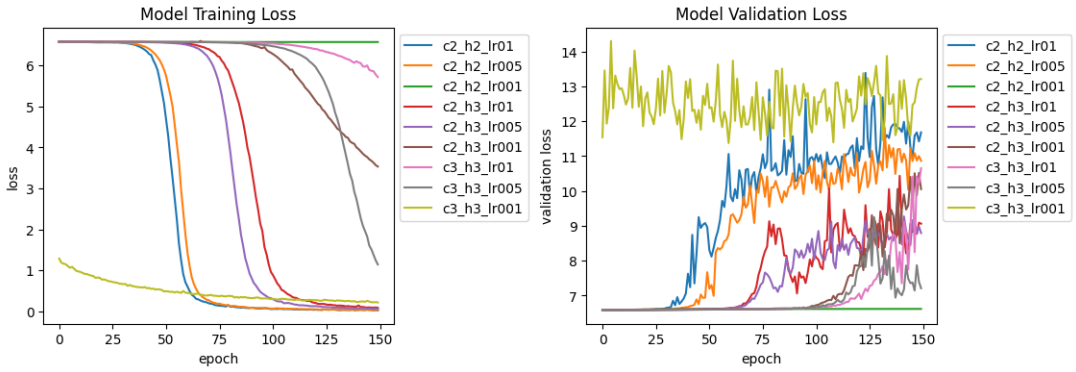


Figure 4: Plot of the losses for the classification models we tested

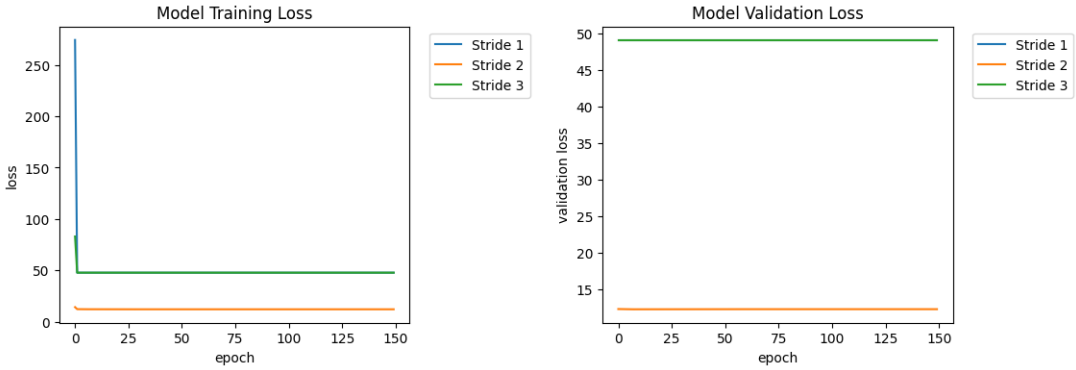


Figure 5: Graph of the loss of our regression models

expectations of the multi head models and we were proven both right and wrong. The two headed classification model performed great with a common sense error of about 27 minutes and the two headed regression model performed very badly. Our current theory for the great improvement of the two headed classification and the two headed regression models is that the biggest problem for our simple classifier was the very vast number of classes compared to the size of the dataset, 18,000 images which is less than 25 examples per class. Our two headed classification model had 1500 and 300 examples per class, which could explain the difference in performances.

6. The architectures that performed best by far were the two label transformation architectures. The main goal of our experiment here was to see whether a higher number of heads could lead to more precise results. The two head model (predicts only one clock hand) performed clearly better, with a common sense error of about 20 minutes on the 150 by 150 dataset. This is proof for us, that the performance of the two headed classification model wasn't inherently due to the increase in number of heads and more likely because of the data issue we mentioned. Although, looking at figure 6 the 4 headed model seemed to still be converging, implying that it might converge slower but to a better result than what we observed.

7. Finally, the following table contains the results on all the models we tested on the large dataset for this task (Head count here is per head).

8. We chose to attempt modifications of the two headed cos and sine model and the two headed

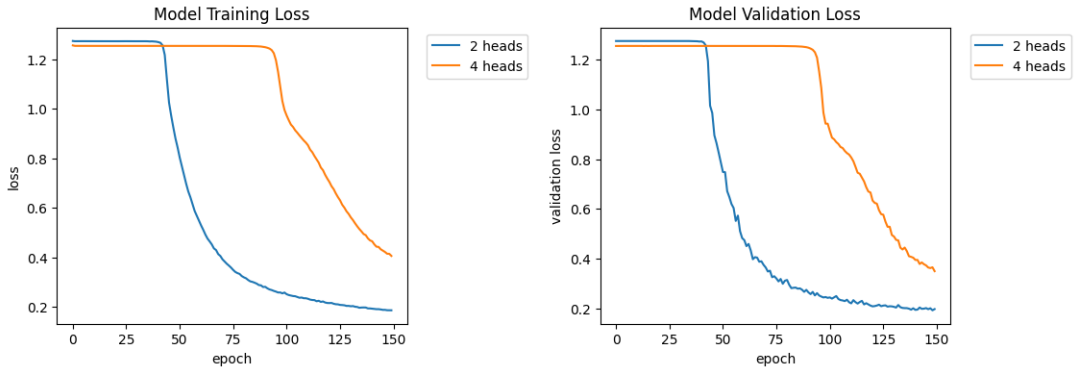


Figure 6: Plot of the losses for the label transformation models we tested

Model	Convolution Patterns	Hidden Layers	Stride	Learning Rate	Head Count	Minute Type	Hour Type	Common Sense Error
Classification	2	2	1x1	0.1	1	N/A	N/A	2:14
Classification	2	2	1x1	0.05	1	N/A	N/A	2:14
Classification	2	2	1x1	0.01	1	N/A	N/A	2:14
Classification	2	3	1x1	0.1	1	N/A	N/A	2:14
Classification	2	3	1x1	0.05	1	N/A	N/A	2:14
Classification	2	3	1x1	0.01	1	N/A	N/A	2:14
Classification	3	3	1x1	0.1	1	N/A	N/A	2:14
Classification	3	3	1x1	0.05	1	N/A	N/A	2:14
Classification	3	3	1x1	0.01	1	N/A	N/A	2:14
Regression	2	3	1x1	0.01	1	N/A	N/A	2:14
Regression	2	3	2x2	0.01	1	N/A	N/A	2:14
Regression	2	3	2x1 and 1x2	0.01	1	N/A	N/A	1:29
MultiHead	2	3	1x1	0.01	2	Classification	Regression	1:36
MultiHead	2	3	1x1	0.01	2	Classification	Classification	0:27
MultiHead	2	3	1x1	0.01	2	Regression	Regression	2:14
MultiHead	2	3	1x1	0.01	2	Regression	Classification	2:00
Label Transform	2	4	1x1	0.01	4	Regression	Regression	1:20
Label Transform	2	3	1x1	0.01	2	N/A	Regression	0:20

Figure 7: Table of our results in task 2.1

classification model. All five models were great improvements in accuracy. The mean common sense errors they achieved are in order 16.7, 11.54, 11.41 and 7.72 minutes. Using a layer of max pooling feels somewhat like a workaround but does produce quite impressive results. We note that both of the other methods tested simply make the network deeper which makes us wonder until when that would increase performances. Sadly, we cannot currently test this.

9. As shown in figure 8, it is quite impressive how fast the label transformation models converged, although it did finish with a higher error.

2.4 Conclusion

1. Main takeaways:
   (a) Label transformation can allow for great results. Label transformations are a technique we did not have any experience with but these experiments proved its power. The two headed label transformation model was one of the simplest ones we tested, yet its performance was

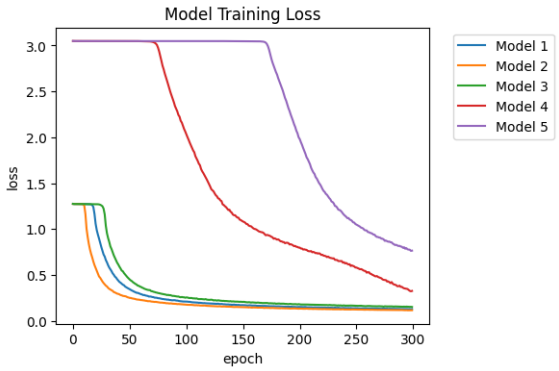


Figure 8: Graph of our final models' loss

impressive.

(b) This final test was a good reminder of something mentioned in lecture, when we make the network deeper by adding convolutions or layers it usually performs better. And deepening the layer is what most techniques here did to great effect.

(c) Neither of us had experience with multi head networks. These experiments have intrigued us to the possibilities of these networks when applied to the right problem. They seem to be able to greatly improve the accuracy of networks that don't have much examples per category by making each example count for multiple heads. In that way they could almost be seen as data augmentation tools.

2. Directions for future work

(a) As mentioned previously, testing these architectures trained on a loss function closer to the common sense error would be an interesting experiment. We surmise a difference in performance would exist but wouldn't be very large for most models.

(b) As was specified earlier, the architecture we based ourselves on is one we found on the keras github. If not for time constraints we would have tested more patterns for convolution and its effects combined with other parameters.

(c) Another hyperparameter we wish we could have tested on more is the optimizer of the model. For all models we used stochastic gradient descent with no momentum as it is the most "vanilla" optimizer. But we know that using ADAM or other optimizers with momentum, decay and other parameters could have yielded interesting results.

(d) The last part we wish we could have experimented more is the label transformations. Even for the regression model there are multiple transformations that are valid like the number of minutes passed since 12 or the number of hours passed since 12 but there are many more depending on the model and we couldn't experiment as much with it as may have been ideal.