COMPUTER VISION
Assignment 5 Report (Training Arc)

Samartha S M

2018101094

Introduction

Deep Learning (DL) (also known as **Deep Structured Learning**) is part of broader **Machine Learning (ML)** based on **Artificial Neural Networks (ANNs)** with representation learning. **Convolutional Neural Networks (CNNs)** are a class of **Deep Learning Neural Networks**. They're very useful for image classification and recognition. We are using **Convolutional Neural Networks (CNNs)** for the purpose of the Image Classification problem as part of assignment 5. The framework used for this purpose is **PyTorch**. The main parts of the code were writing data loaders, model and training code. This report contains the study of various parametric choices of the **Deep Learning (DL)** model architecture design.

Baseline CNN Model Architecture

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
  )
)
```

The images are transformed as follows,

- Resized to (64, 64)
- Converted to tensors using **ToTensor()**

The dataset given is then split into the training set and validation set in the ratio of **90:10**. A **Dataloader** is used to create batches of training data and validation data with a batch size of **32**.

The baseline **CNN** model has 10 layers in total, out of which 2 are Convolutional layers, 2 are Max Pool layers, 3 are ReLu activation layers, 2 are Full Connected Linear layers and 1 is flattening layer. For both Convolutional layers, the kernel size is (3, 3), the stride is (1, 1) and padding applied is (1, 1). The ReLU activation layers are used after each Convolutional layer and Fully Connected Linear layer to introduce non-linearity.

Hyperparameters

- **Number of Epochs** = 10
- **Learning rate** = 0.001
- **Batch size** = 64
- **Number of conv layers** = 2
- **Number of fully connected (FC) linear layers** = 2
- **Dropout** = 0

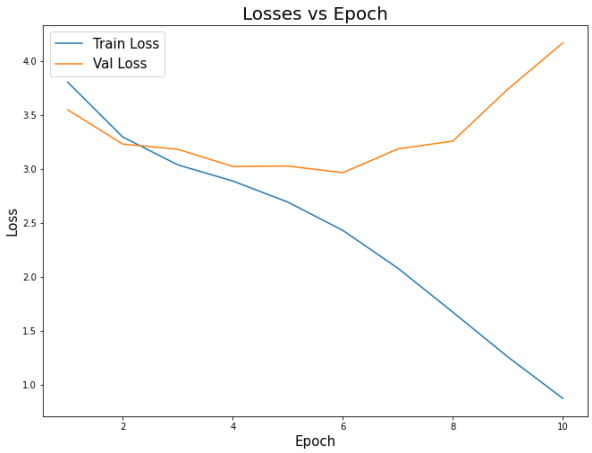
The optimizer used for training is **Adam**

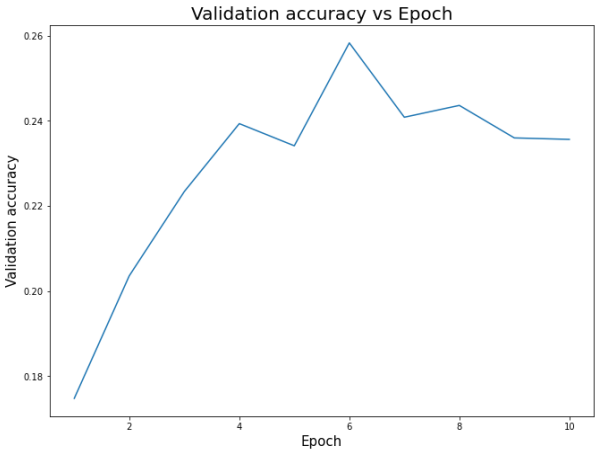
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.8009023666381836  Val loss:  3.5452396869659424   Val acc: 0.17476850748062134
Epoch: 2        Train loss: 3.293173313140869   Val loss:  3.2278521060943604   Val acc: 0.20358796417713165
Epoch: 3        Train loss: 3.0353925228118896  Val loss:  3.179802417755127    Val acc: 0.22337962687015533
Epoch: 4        Train loss: 2.885687828063965   Val loss:  3.02030086517334     Val acc: 0.23935183882713318
Epoch: 5        Train loss: 2.690101385116577   Val loss:  3.0244717597961426   Val acc: 0.23414351046085358
Epoch: 6        Train loss: 2.4279983043670654  Val loss:  2.962477445602417    Val acc: 0.25833332538604736
Epoch: 7        Train loss: 2.0780863761901855  Val loss:  3.1826560497283936   Val acc: 0.24085648357868195
Epoch: 8        Train loss: 1.6704670190811157  Val loss:  3.2556653022766113   Val acc: 0.24363425374031067
Epoch: 9        Train loss: 1.2552039623260498  Val loss:  3.736119270324707    Val acc: 0.2359953671693802
Epoch: 10       Train loss: 0.8708454370498657  Val loss:  4.165708065032959    Val acc: 0.23564815521240234
```

Here, we can see the training loss, validation loss and validation accuracy at each epoch. From the image, we can see that **validation loss** has attained a minimum value of **2.9625** at epoch **6**. The **validation accuracy** has attained a maximum value of **25.83%** at epoch **6**.

Graph Plots





We can see from the above graphs and loss and accuracy data that the training is going on well as the training loss decreases after each epoch. However, we can see that the validation error decreases at first and then starts to increase from the 7th epoch. This tells us that the model is overfitting to training data from the 7th epoch. Hence, we need to do early stopping at the 6th epoch of training to get a well-generalized model. Parallelly we can see that the validation accuracy increases at first and then decreases. Hence, the sweet spot for the number of epochs for this data with this baseline model is 6. This can vary based on the model architecture. As we are using a shallow network (not soo deep) here, the training is quicker and learns quickly in just 6 epochs (i.e. well-generalized model without overfitting). In further sections, we'll be studying various model architectures with different parameters where we'll be using deep networks too. There we can expect to see that number of epochs for training increases. The total time taken by the classification model to train on the training data is 14 minutes 19 seconds.

Analysis of various parameters

Now, we'll study different parameters and analyse how they affect image classification.

Batch Norm

Training deep neural networks can be challenging because the distribution of inputs to deep layers can change after each batch because of weights getting updated. This is called

the **internal covariate shift**. Hence, batch normalization is a technique to overcome this problem that standardizes the inputs to a layer for each batch. Hence, the distribution of inputs remains intact for each batch. Hence, this can accelerate the training phase and can even act as a regularization technique reducing the generalization error.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): BatchNorm1d(65536, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): Linear(in_features=65536, out_features=512, bias=True)
    (11): ReLU()
    (12): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): Linear(in_features=512, out_features=62, bias=True)
  )
)
```

We can see that before convolutional layers and linear layers, a batch normalization layer is added to normalize the inputs to those layers. As the inputs for convolutional layers is 2D, we use BatchNorm2d to normalize the 2D inputs. Similarly, for linear layers whose inputs are 1D, we use BatchNorm1D to normalize the 1D inputs. The rest of the layers are similar to the ones we used in the baseline model.

Hyperparameters

- **Number of Epochs** = 10
- **Learning rate** = 0.001
- **Batch size** = 64
- **Number of conv layers** = 2
- **Number of fully connected (FC) linear layers** = 2
- **Dropout** = 0

The optimizer used for training is **Adam**

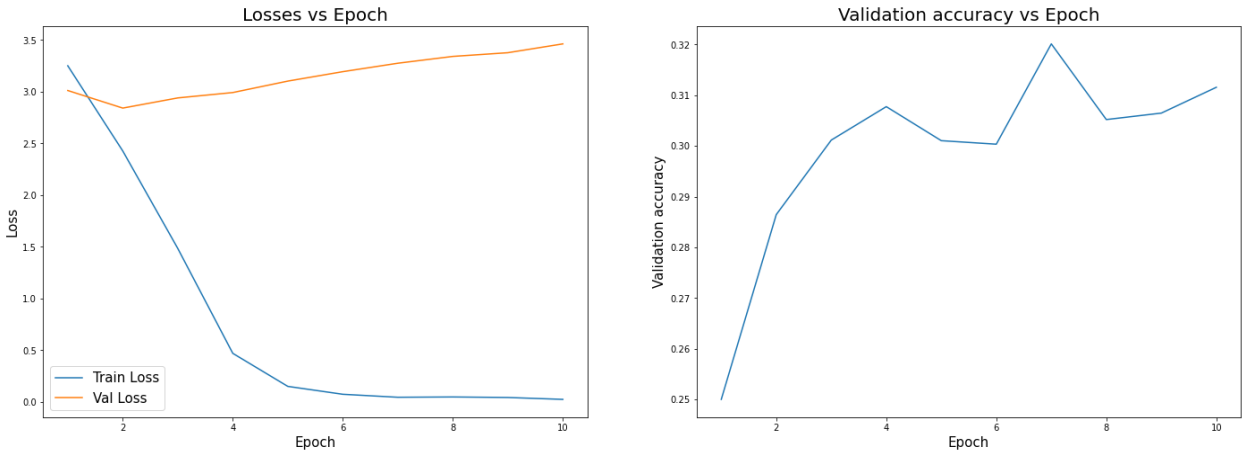
Inputs to each layer are batch normalized

Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.2514564990997314          Val loss:  3.012204170227051    Val acc: 0.25
Epoch: 2        Train loss: 2.4273276329040527          Val loss:  2.842170238494873    Val acc: 0.2864583432674408
Epoch: 3        Train loss: 1.484918236732483           Val loss:  2.940523862838745    Val acc: 0.3011574149131775
Epoch: 4        Train loss: 0.4707016348838806          Val loss:  2.9925129413604736   Val acc: 0.30775463581085205
Epoch: 5        Train loss: 0.1513270139694214          Val loss:  3.1031506061553955   Val acc: 0.3010416626930237
Epoch: 6        Train loss: 0.07494168728590012         Val loss:  3.1943564414978027   Val acc: 0.3003472089767456
Epoch: 7        Train loss: 0.046070754528045654        Val loss:  3.275946617126465    Val acc: 0.3201388716697693
Epoch: 8        Train loss: 0.048966195434331894        Val loss:  3.3419060707092285   Val acc: 0.30520832538604736
Epoch: 9        Train loss: 0.04391368478536606         Val loss:  3.377659797668457    Val acc: 0.3064814805984497
Epoch: 10       Train loss: 0.02558758109807968         Val loss:  3.4630534648895264   Val acc: 0.3115740716457367
```

Here, we can see that the training loss is decreasing quickly compared to the baseline model even with the same learning rate of 0.001. We can even see the maximum attained validation accuracy of **32.01%** is also high compared to the baseline model and it attains faster (in fewer epochs). The minimum validation loss attained is **2.842**.

Graph Plots



Here, we can see that the plot of train loss is decreasing steeply and saturates after the 6th epoch. Hence, the learning process is faster when compared to the baseline model. We can even see that the validation loss attains minimum at the 2nd epoch itself and starts to increase gradually. However, the accuracy keeps increasing even after the 2nd epoch. Hence, from the validation loss plot, we can say that the model starts overfitting the training data from 3rd epoch. But from the validation accuracy plot, it seems that the model starts overfitting the training data from roughly the 5th epoch because the validation accuracy takes a sudden dip at the 5th epoch. Even though it increases at peaks at the 7th epoch, it might be by chance. Hence, batch normalization quickens the learning process.

Adding new layers

Now, we'll look at how adding new layers to the model affects the training process. Adding new layers to the model means making the model deeper. Hence, the learning time can increase. This can increase the accuracy due to better learning. Making networks deeper helps in better classification because the model learns the underlying patterns better in deeper networks than in shallow networks. As the number of layers increases, the classifier function becomes more complex, hence building the ability to learn highly complex function. Hence, with sufficient data, the model can learn complex function to classify images with high accuracy.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Flatten(start_dim=1, end_dim=-1)
    (13): Linear(in_features=16384, out_features=1024, bias=True)
    (14): ReLU()
    (15): Linear(in_features=1024, out_features=512, bias=True)
    (16): ReLU()
    (17): Linear(in_features=512, out_features=256, bias=True)
    (18): ReLU()
    (19): Linear(in_features=256, out_features=62, bias=True)
  )
)
```

Above is the model architecture of the deep network with new layers added. However, batch normalization layers are removed to understand the effect of the addition of new layers better by comparing to the baseline model. 2 more convolutional layers are added, hence, 2 more ReLU activation layers and MaxPool layers are added. 2 more Linear layers are added for better learning at the end, hence 2 more ReLU activation layers to introduce non-linearity are added.

Hyperparameters

- **Number of Epochs** = 20
- **Learning rate** = 0.001
- **Batch size** = 64
- **Number of conv layers** = 4
- **Number of fully connected (FC) linear layers** = 4
- **Dropout** = 0

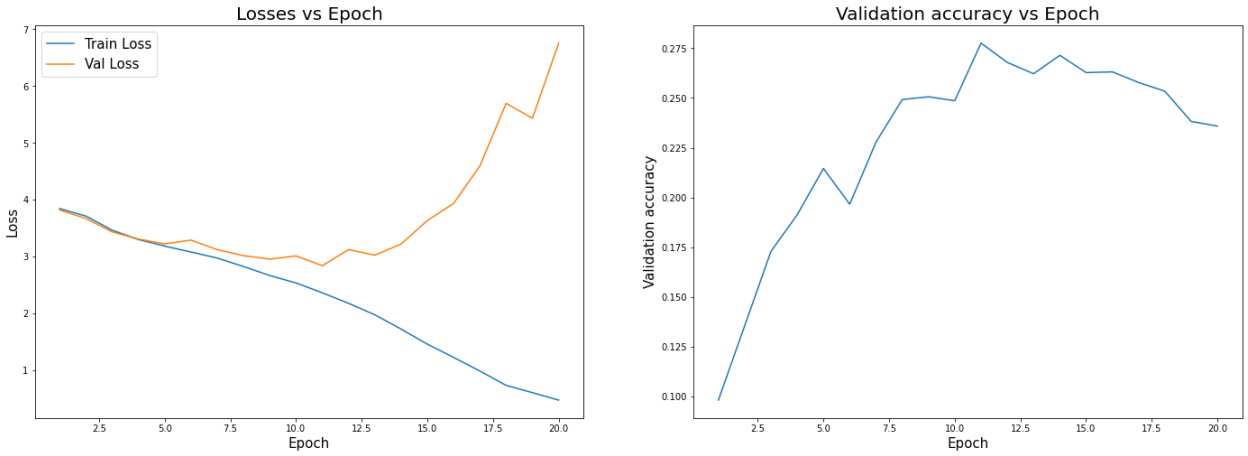
The optimizer used for training is **Adam**

Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.8450374603271484  Val loss:  3.8195338249206543   Val acc: 0.09826388210058212
Epoch: 2        Train loss: 3.712794542312622   Val loss:  3.670804500579834    Val acc: 0.135532408952713
Epoch: 3        Train loss: 3.4650380611419678  Val loss:  3.4345204830169678   Val acc: 0.17291666567325592
Epoch: 4        Train loss: 3.2995402812957764  Val loss:  3.3094592094421387   Val acc: 0.19131943583488464
Epoch: 5        Train loss: 3.1843953132629395  Val loss:  3.223191261291504    Val acc: 0.21458333730697632
Epoch: 6        Train loss: 3.080838680267334   Val loss:  3.2902259826660156   Val acc: 0.19664351642131805
Epoch: 7        Train loss: 2.9746694564819336  Val loss:  3.1223511695861816   Val acc: 0.22777776420116425
Epoch: 8        Train loss: 2.8254506587982178  Val loss:  3.016472101211548    Val acc: 0.2491898238658905
Epoch: 9        Train loss: 2.667064905166626   Val loss:  2.9555089473724365   Val acc: 0.25057870149612427
Epoch: 10       Train loss: 2.5363616943359375  Val loss:  3.012295722961426    Val acc: 0.24861110746860504
Epoch: 11       Train loss: 2.363375186920166   Val loss:  2.8392457962036133   Val acc: 0.27754631638526917
Epoch: 12       Train loss: 2.1782913208007812  Val loss:  3.124391794204712    Val acc: 0.26782408356666565
Epoch: 13       Train loss: 1.9759843349456787  Val loss:  3.0241293907165527   Val acc: 0.262152761220932
Epoch: 14       Train loss: 1.7251962423324585  Val loss:  3.221345901489258    Val acc: 0.2714120149612427
Epoch: 15       Train loss: 1.4594955444335938  Val loss:  3.6341662406921387   Val acc: 0.26273149251937866
Epoch: 16       Train loss: 1.2263879776000977  Val loss:  3.9355721473693848   Val acc: 0.2630787193775177
Epoch: 17       Train loss: 0.9865505695343018  Val loss:  4.5945820808410645   Val acc: 0.2577546238899231
Epoch: 18       Train loss: 0.7342895865440369  Val loss:  5.69471549987793     Val acc: 0.2533564865589142
Epoch: 19       Train loss: 0.6066657900810242  Val loss:  5.433778285980225    Val acc: 0.23819443583488464
Epoch: 20       Train loss: 0.474944144487381   Val loss:  6.756147384643555    Val acc: 0.23587962985038757
```

I have trained the model for 20 epochs because the model didn't seem to overfit in the first 10 iterations. Here, we can see that the maximum attained validation accuracy is **27.75%** at 11^{th} epoch. We can see that this validation accuracy is more than that of the baseline model. The minimum attained validation loss is **2.839** at 11^{th} epoch which is lesser than that of the baseline model. Hence, we can see that adding more layers resulted in better training. However, we can see that the number of epochs to reach similar training loss as that of the baseline model is almost doubled. It took **5** epochs to reach training loss of **2.69** in baseline model, but this new model took **9** epochs to reach training loss of **2.667**. Hence, its evident from the loss and accuracy values that the training rate has been reduced by half in terms of number of epochs. The learning rate is still the same as that of the baseline model which is **0.001**.

Graph Plots



From the graphs, we can see that the validation loss decreases first till the **11th** epoch where it attains minimum and then starts to increase. Even the validation accuracy increases first till **11th** epoch where it peaks and then starts to decrease. Hence, its evident that the model starts to overfit the training data from the **12th** epoch. So, comparing this to the baseline model, we can clearly state that the training rate has been nearly halved (i.e number of epochs required to train model is almost doubled). It took 6 epochs for the model to start overfitting, but this model took **11** epochs (which is almost double of 6). Hence, by doubling the number of convolutional and fully connected linear layers, the training phase has doubled to get a good generalized model. But as we saw the values, the accuracy had increased a bit (by almost **2%**) because of learning complex functions from the data.

Dropout

Dropout is another regularization method used in neural networks. Dropout simply means ignoring a few nodes in the network (not considered in both forward or backward pass). Hence, at each epoch for each batch, few nodes are probabilistically dropped out. Most of the parameters are occupied in fully connected linear layers (which is generally present at the end in CNN). Over training time, these neurons develop co-dependency amongst each other, hence curbing the individual power of each neuron. Which leads to overfitting. Hence, few neurons are dropped out in each training batch to reduce the interdependent learning amongst the individual neurons.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): Dropout(p=0.4, inplace=False)
    (9): ReLU()
    (10): Linear(in_features=512, out_features=62, bias=True)
  )
)
```

We can see that **1 dropout** layer is added right after the first fully connected linear layer with drop probability being **0.4**. Hence, each of the **512** outputs from the first fully connected linear layer will be dropped with probability of **0.4**. Rest all layers are exactly similar to those of the baseline model. The dropout layer is added only after fully connected linear layers (except for the last fully connected linear layer). This is because dropout makes more sense for the fully connected linear layers, which have dense mapping between neurons of adjacent layers. In convolutional layers, it doesn't make sense to add dropout layers because the patch (convolution filter) we use has to scan through all pixels to obtain proper information or patterns in the input. If dropout layers are added to convolutional layers, then the underlying patterns would not be correctly recognized and hence lead to bad learning.

Hyperparameters

- **Number of Epochs** = 20
- **Learning rate** = 0.001
- **Batch size** = 64
- **Number of conv layers** = 2
- **Number of fully connected (FC) linear layers** = 2
- **Dropout** = 0.4

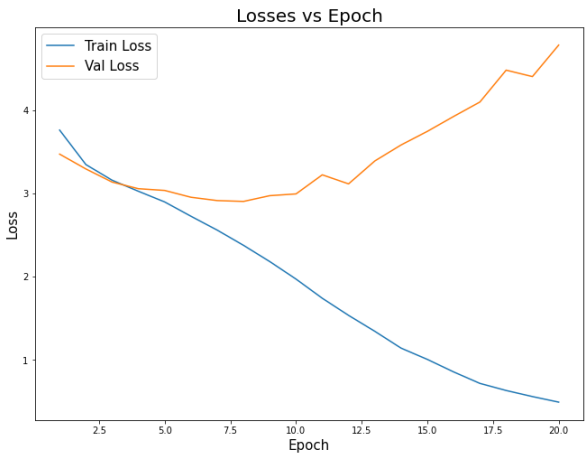
The optimizer used for training is **Adam**

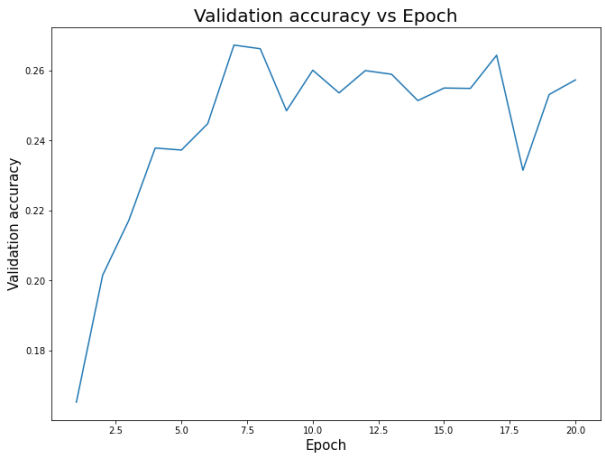
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.7588577270507812  Val loss:  3.4702868461608887   Val acc: 0.16527776420116425
Epoch: 2        Train loss: 3.3449056148529053  Val loss:  3.2922985553741455   Val acc: 0.2015046328306198
Epoch: 3        Train loss: 3.1567771434783936  Val loss:  3.134474039077759    Val acc: 0.21724537014961243
Epoch: 4        Train loss: 3.024395704269409   Val loss:  3.0552663803100586   Val acc: 0.237847238779068
Epoch: 5        Train loss: 2.896204710006714   Val loss:  3.0342774391174316   Val acc: 0.23726850748062134
Epoch: 6        Train loss: 2.7245516777038574  Val loss:  2.9525108337402344   Val acc: 0.2447916716337204
Epoch: 7        Train loss: 2.5579094886779785  Val loss:  2.9126439094543457   Val acc: 0.267245352268219
Epoch: 8        Train loss: 2.3754780292510986  Val loss:  2.9024837017059326   Val acc: 0.26620370149612427
Epoch: 9        Train loss: 2.1804094314575195  Val loss:  2.97159743309021     Val acc: 0.24849537014961243
Epoch: 10       Train loss: 1.9700695276260376  Val loss:  2.992760181427002    Val acc: 0.26006942987442017
Epoch: 11       Train loss: 1.739863395690918   Val loss:  3.2219367027282715   Val acc: 0.2535879611968994
Epoch: 12       Train loss: 1.5344973802566528  Val loss:  3.113173484802246    Val acc: 0.25995370745658875
Epoch: 13       Train loss: 1.3431211709976196  Val loss:  3.3884048461914062   Val acc: 0.25891202688217163
Epoch: 14       Train loss: 1.1410062313079834  Val loss:  3.5805020332336426   Val acc: 0.25138890743255615
Epoch: 15       Train loss: 1.0057811737060547  Val loss:  3.7438621520996094   Val acc: 0.2549768388271332
Epoch: 16       Train loss: 0.855600893497467   Val loss:  3.9227209091186523   Val acc: 0.25486108660697937
Epoch: 17       Train loss: 0.7178280353546143  Val loss:  4.095944881439209    Val acc: 0.26435184478759766
Epoch: 18       Train loss: 0.6325216889381409  Val loss:  4.478621482849121    Val acc: 0.23148149251937866
Epoch: 19       Train loss: 0.559575080871582   Val loss:  4.401404857635498    Val acc: 0.25312501192092896
Epoch: 20       Train loss: 0.49275797605514526 Val loss:  4.7797722816467285   Val acc: 0.25729167461395264
```

We can see from the values of losses and accuracies itself that dropout has prevented the model from overfitting. We can see that the minimum validation loss attained is **2.902** which is lesser than that of the baseline model. The maximum validation accuracy attained is **26.72%** which is higher than that of the baseline model. As we saw in the baseline model, the validation accuracy started decreasing after a few epochs indicating overfitting of the model to the training data. But this model seems to be stable around **25.5%** validation accuracy even at 20th epoch. Hence, its clearly evident that dropout prevents overfitting to some extent and hence acts as a regularization technique by reducing the interdependent learning amongst the neurons.

Graph Plots





From the graph plots, we can see that the training loss plot is in general decreasing with epochs, the validation loss decreases first and then starts to increase gradually. But the validation accuracy plot shows that the accuracy is stable around **25.5%**. It might have come down to **23.15%** at 18th epoch, but it has come back up in the next epoch. Hence, it seems to be stable and hence shows that the model has overcome overfitting to some extent. As we have just one dropout layer, the effect might not be as clear, but when we will have more dropout layers, then the effect would be clearly visible through loss and accuracy plots.

Different activation functions at the end

Now let's study what happens when we keep different activation functions at the end. Activation functions are used to introduce non linearity into the hidden layers because if non linearity is not added, then the whole model can be reduced to a single neuron with a linear function. Then it won't be able to learn complex functions.

Sigmoid activation function

Sigmoid function is also called a logistic function. Range of output is (0, 1). It has an S-shaped curve. It's a popular activation function for neural networks.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
    (10): Sigmoid()
  )
)
```

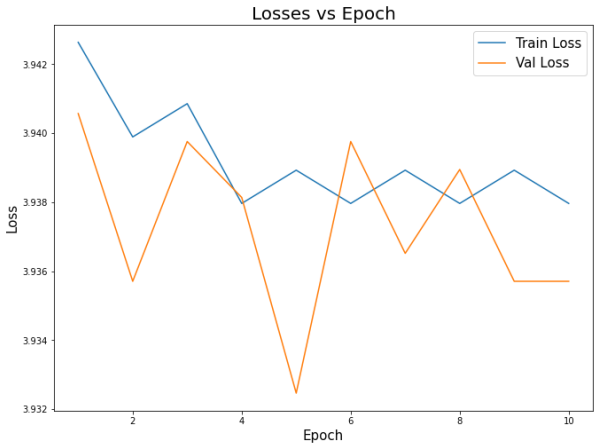
I have added a sigmoid activation layer at the end. Rest all 10 layers are the same as those of the baseline model.

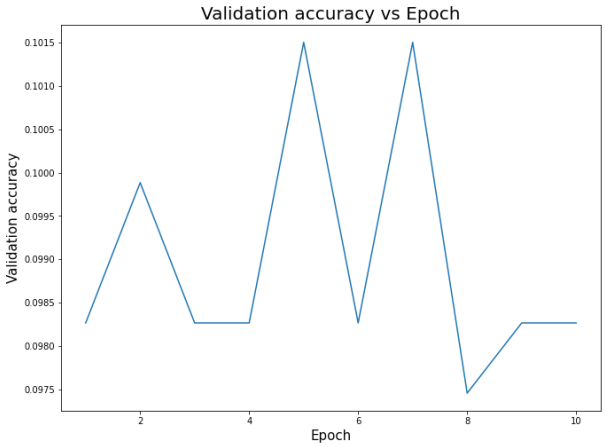
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.9426326751708984  Val loss:  3.9405677318573     Val acc: 0.09826388210058212
Epoch: 2        Train loss: 3.9398906230926514  Val loss:  3.935706615447998   Val acc: 0.09988425672054291
Epoch: 3        Train loss: 3.940854787826538   Val loss:  3.9397575855255127  Val acc: 0.09826388210058212
Epoch: 4        Train loss: 3.9379634857177734  Val loss:  3.9381370544433594  Val acc: 0.09826388210058212
Epoch: 5        Train loss: 3.938926935195923   Val loss:  3.9324657917022705  Val acc: 0.10150463134050369
Epoch: 6        Train loss: 3.9379634857177734  Val loss:  3.9397575855255127  Val acc: 0.09826388210058212
Epoch: 7        Train loss: 3.938926935195923   Val loss:  3.936516761779785   Val acc: 0.10150463134050369
Epoch: 8        Train loss: 3.9379634857177734  Val loss:  3.9389472007751465  Val acc: 0.09745370596647263
Epoch: 9        Train loss: 3.938926935195923   Val loss:  3.935706853866577   Val acc: 0.09826388210058212
Epoch: 10       Train loss: 3.9379634857177734  Val loss:  3.935706615447998   Val acc: 0.09826388210058212
```

We can see that the loss values (both train and validation) don't change a lot. Both losses are around **3.93** for all epochs. Even the validation accuracy is not changing a lot. It is around **9.8%** for all epochs. Hence, the model is not learning the underlying complex function properly.

Graph Plots





Even from the plots, we can see that the values are not changing a lot. They seem to oscillate a lot, but that is due to the scale of the loss axis. Hence, it is evident from graphs that the model is not learning the complex function required to classify images. This might be because of the output range of sigmoid activation function. As its range is (0, 1), the output values are very low w.r.t the loss function. Hence, the loss values will be almost the same for all epochs as the probability distribution between different classes will be similar in all epochs. Hence, the optimizer is not able to optimize the model through loss function.

LogSigmoid activation function

This activation function is a log of sigmoid activation functions.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
    (10): LogSigmoid()
  )
)
```

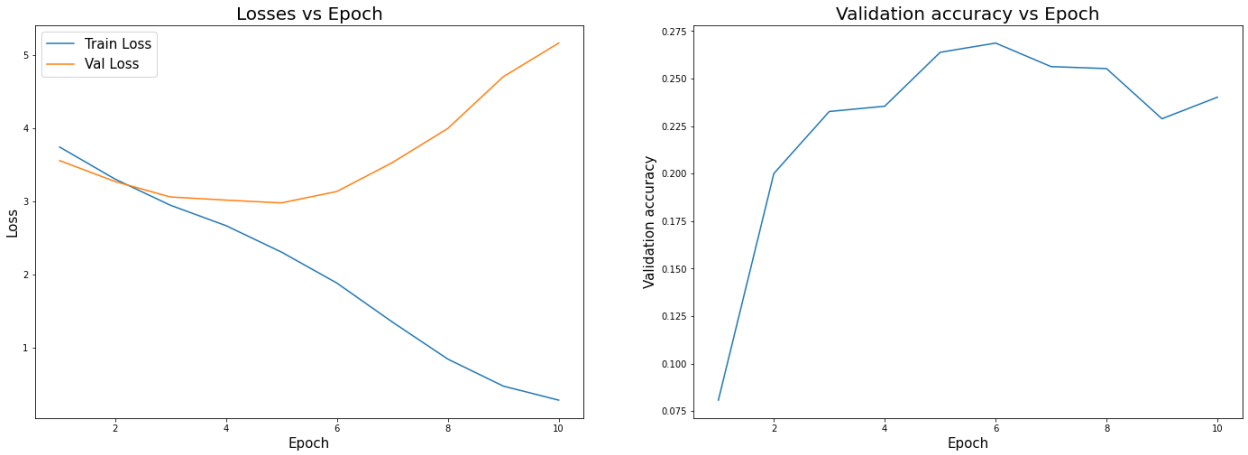
I have added the LogSigmoid activation function layer at the end of the model. Rest all layers are the same as that of the baseline model.

Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.739628314971924   Val loss:  3.5548691749572754   Val acc: 0.0807870402932167
Epoch: 2        Train loss: 3.3021059036254883  Val loss:  3.268944263458252    Val acc: 0.20000000298023224
Epoch: 3        Train loss: 2.9448506832122803  Val loss:  3.0579240322113037   Val acc: 0.2326388955116272
Epoch: 4        Train loss: 2.667083263397217   Val loss:  3.0154972076416016   Val acc: 0.23541666567325592
Epoch: 5        Train loss: 2.3064627647399902  Val loss:  2.9773666858673096   Val acc: 0.2637731432914734
Epoch: 6        Train loss: 1.8835482597351074  Val loss:  3.1333975791931152   Val acc: 0.26863425970077515
Epoch: 7        Train loss: 1.3520647287368774  Val loss:  3.527575969696045    Val acc: 0.2562499940395355
Epoch: 8        Train loss: 0.8465397953987122  Val loss:  3.9931178092956543   Val acc: 0.2552083432674408
Epoch: 9        Train loss: 0.4778692424297333  Val loss:  4.698664665222168    Val acc: 0.22881942987442017
Epoch: 10       Train loss: 0.2862509787082672  Val loss:  5.158344745635986    Val acc: 0.24016204476356506
```

Now, we can see that the loss values are changing and hence the model is training properly. The minimum validation loss attained is **2.977** at **5th** epoch. The maximum validation accuracy attained is **26.86%** at **6th** epoch.

Graph Plots



We can see that the training is going well. The model seems to overfit at **5th** epoch, when it attains minimum validation loss. The minimum validation loss attained is **2.977**. The maximum validation accuracy attained is **26.86%** at **6th** epoch. The model tends to be overfitting the training data from **7th** epoch as can be seen from the above plots.

Softmax activation function

Softmax is another popular activation function for neural networks.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
    (10): Softmax(dim=None)
  )
)
```

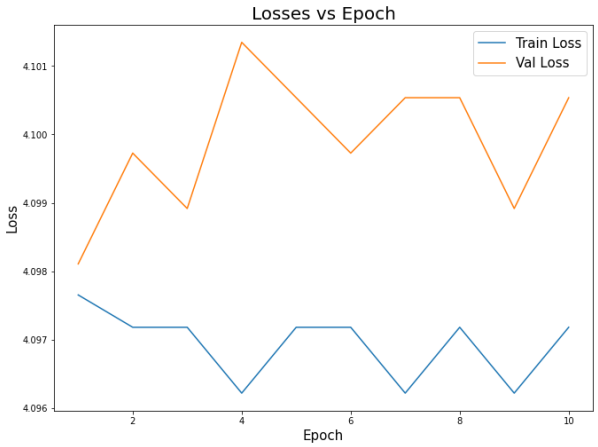
I have added a softmax activation layer at the end. Rest all layers are same as that of the baseline model.

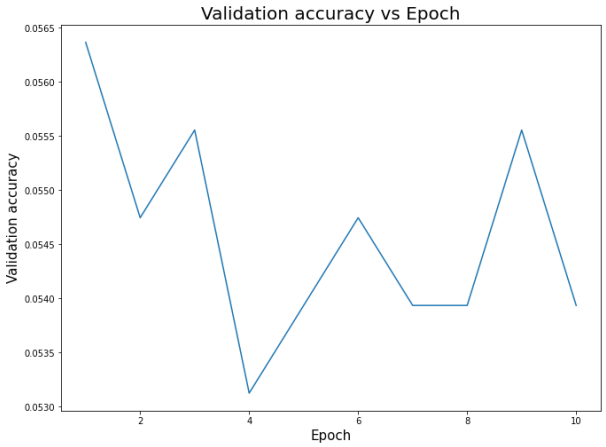
Loss and Accuracy Values

```
Epoch: 1        Train loss: 4.097652435302734   Val loss:  4.098106384277344    Val acc: 0.056365739554166794
Epoch: 2        Train loss: 4.097179889678955   Val loss:  4.099726676940918    Val acc: 0.054745372384786606
Epoch: 3        Train loss: 4.097179889678955   Val loss:  4.098916530609131    Val acc: 0.0555555559694767
Epoch: 4        Train loss: 4.096216678619385   Val loss:  4.101346969604492    Val acc: 0.05312500149011612
Epoch: 5        Train loss: 4.097179889678955   Val loss:  4.100536823272705    Val acc: 0.05393518507480621
Epoch: 6        Train loss: 4.097179889678955   Val loss:  4.099726676940918    Val acc: 0.054745372384786606
Epoch: 7        Train loss: 4.096216678619385   Val loss:  4.100536823272705    Val acc: 0.05393518507480621
Epoch: 8        Train loss: 4.097179889678955   Val loss:  4.100536823272705    Val acc: 0.05393518507480621
Epoch: 9        Train loss: 4.096216678619385   Val loss:  4.098916530609131    Val acc: 0.055555559694767
Epoch: 10       Train loss: 4.097179889678955   Val loss:  4.100536823272705    Val acc: 0.05393518507480621
```

Again we can see that the values of losses and accuracies are inconsistent. The values are almost constant with little oscillation for all epochs. Hence, the model is not training properly. As discussed earlier with sigmoid activation function, it might be because of the output range of softmax activation function.

Graph Plots





Even from the plots, we can see that the values are not changing a lot. They seem to oscillate a lot, but that is due to the scale of the loss axis. Hence, it is evident from graphs that the model is not learning the complex function required to classify images. This might be because of the output range of the softmax activation function. As its range is (0, 1), the output values are very low w.r.t the loss function. Hence, the loss values will be almost the same for all epochs as the probability distribution between different classes will be similar in all epochs. Hence, the optimizer is not able to optimize the model through loss function.

LogSoftmax activation function

This activation function is a log of softmax activation functions.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
    (10): LogSoftmax(dim=None)
  )
)
```

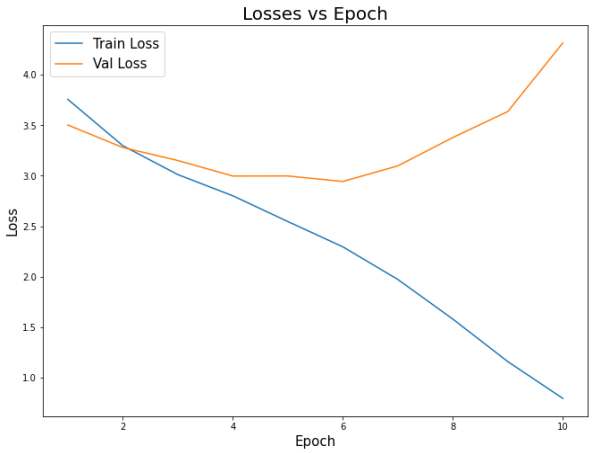
I have added the LogSoftmax activation layer at the end of the model. Rest all layers are the same as that of the baseline model.

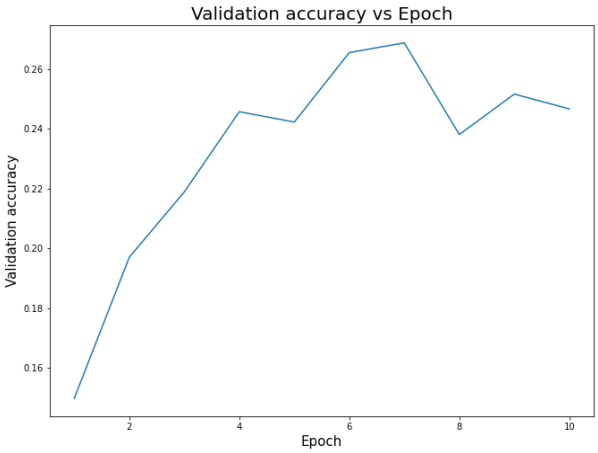
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.757093906402588   Val loss:  3.5025219917297363   Val acc: 0.14976851642131805
Epoch: 2        Train loss: 3.297844648361206   Val loss:  3.2799291610717773   Val acc: 0.1969907432794571
Epoch: 3        Train loss: 3.011723518371582   Val loss:  3.1510562896728516   Val acc: 0.21886573731899261
Epoch: 4        Train loss: 2.8015968799591064  Val loss:  2.9973435401916504   Val acc: 0.2457175850868225
Epoch: 5        Train loss: 2.5465383529663086  Val loss:  2.9979331493377686   Val acc: 0.2422453761100769
Epoch: 6        Train loss: 2.2963738441467285  Val loss:  2.9432268142700195   Val acc: 0.2655092477798462
Epoch: 7        Train loss: 1.9731173515319824  Val loss:  3.0977916717529297   Val acc: 0.26875001192092896
Epoch: 8        Train loss: 1.5806059837341309  Val loss:  3.3784618377685547   Val acc: 0.23807869851589203
Epoch: 9        Train loss: 1.1591668128967285  Val loss:  3.6364803314208984   Val acc: 0.2516203820705414
Epoch: 10       Train loss: 0.7953247427940369  Val loss:  4.315735816955566    Val acc: 0.24664351344108582
```

The values seem to be decreasing. Hence the model is training properly. The minimum attained validation loss is **2.943** at **6th** epoch. The maximum attained validation accuracy is **26.87%** at **7th** epoch.

Graph Plots





The graph plots show that the model is training properly. As we used log, it brings the values out of the range (0, 1). Hence, the optimizer can optimize better on the loss function. The model seems to be overfitting the training data from **8th** epoch.

Tanh activation function

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
    (10): Tanh()
  )
)
```

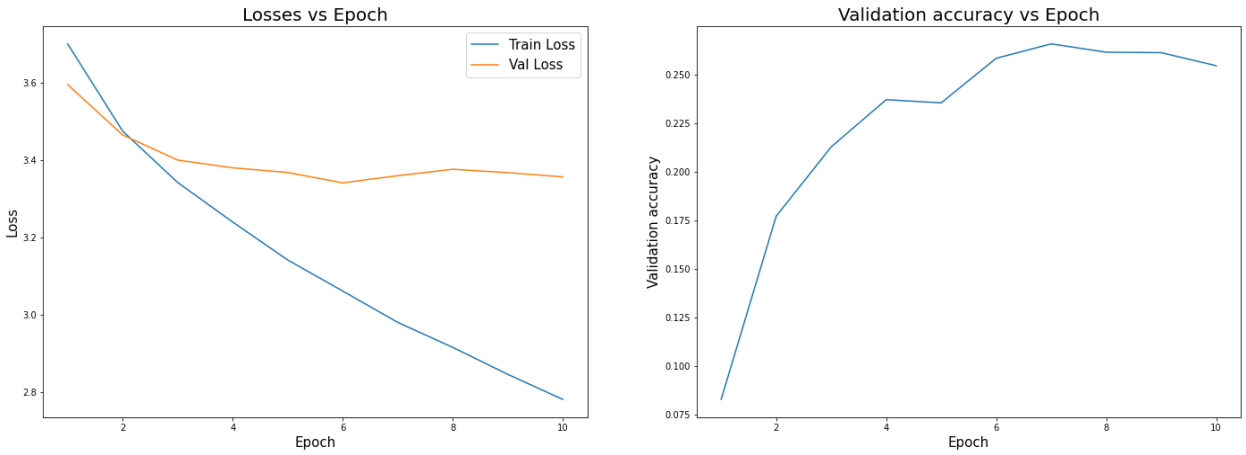
I have added an Tanh activation function layer at the end of the model. Rest all layers are the same as that of the baseline model.

Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.7002906799316406  Val loss:  3.5949606895446777   Val acc: 0.08287037163972855
Epoch: 2        Train loss: 3.4745373725891113  Val loss:  3.46462345123291     Val acc: 0.1770833283662796
Epoch: 3        Train loss: 3.3415639400482178  Val loss:  3.3996472358703613   Val acc: 0.2127314954996109
Epoch: 4        Train loss: 3.2392771244049072  Val loss:  3.3795313835144043   Val acc: 0.2370370328426361
Epoch: 5        Train loss: 3.1411473751068115  Val loss:  3.3674206733703613   Val acc: 0.23541666567325592
Epoch: 6        Train loss: 3.0606746673583984  Val loss:  3.340700387954712    Val acc: 0.25833332538604736
Epoch: 7        Train loss: 2.97961163520813    Val loss:  3.359196186065674    Val acc: 0.2657407522201538
Epoch: 8        Train loss: 2.9149301052093506  Val loss:  3.375831127166748    Val acc: 0.2614583373069763
Epoch: 9        Train loss: 2.8451497554779053  Val loss:  3.367065906524658    Val acc: 0.2612268328666687
Epoch: 10       Train loss: 2.780275344848633   Val loss:  3.3560941219329834   Val acc: 0.2545138895511627
```

The model has been trained properly as the values seem to be changing for each epoch. The minimum obtained validation loss is **3.3407** at **6th** epoch. The maximum attained validation accuracy is **26.57%** at **7th** epoch.

Graph Plots



The loss plot shows that the validation error is almost saturated from **6th** epoch. It hasn't increased. Hence, it might be avoiding overfitting of the model. Even the validation accuracy is not decreasing a lot after the **7th** epoch.

Exponential Linear Unit activation function

This is exponential version of Rectified Linear Unit activation function

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
    (10): ELU(alpha=1.0)
  )
)
```

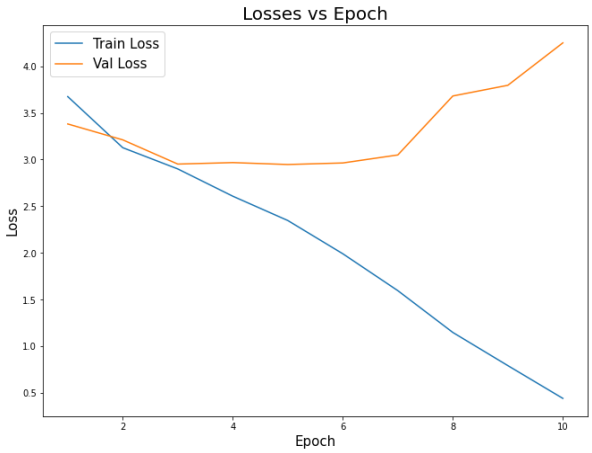
I have added an ELU activation function layer at the end of the model. Rest all layers are the same as that of the baseline model.

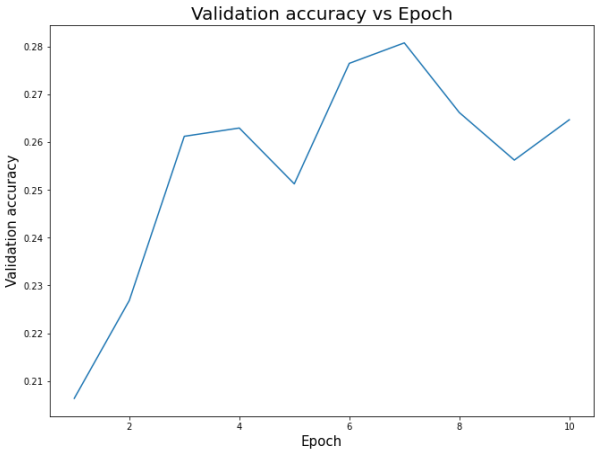
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.67392635345459    Val loss:  3.381145477294922    Val acc: 0.20636574923992157
Epoch: 2        Train loss: 3.126939296722412   Val loss:  3.2107577323913574   Val acc: 0.22685185074806213
Epoch: 3        Train loss: 2.899552822113037   Val loss:  2.951664924621582    Val acc: 0.2612268328666687
Epoch: 4        Train loss: 2.6076202392578125  Val loss:  2.9674627780914307   Val acc: 0.2629629671573639
Epoch: 5        Train loss: 2.3472883701324463  Val loss:  2.9468727111816406   Val acc: 0.25127315521240234
Epoch: 6        Train loss: 1.991278052330017   Val loss:  2.9635159969329834   Val acc: 0.27650463581085205
Epoch: 7        Train loss: 1.596384048461914   Val loss:  3.0492169857025146   Val acc: 0.28078702092170715
Epoch: 8        Train loss: 1.1487948894500732  Val loss:  3.680589437484741    Val acc: 0.26620370149612427
Epoch: 9        Train loss: 0.7937085032463074  Val loss:  3.7950751781463623   Val acc: 0.2562499940395355
Epoch: 10       Train loss: 0.44273459911346436 Val loss:  4.249275207519531    Val acc: 0.2646990716457367
```

The minimum validation loss attained is **2.9516** at **3rd** epoch. The maximum validation accuracy attained is **28.07%** at **7th** epoch.

Graph Plots





The validation loss seems to start increasing from the **4th** epoch. But the validation accuracy reaches maximum at **7th** epoch. But the validation loss is almost constant from **4th** epoch to **7th** epoch. Hence, the model starts to overfit the training data from the 8th epoch.

Rectified Linear Unit activation function

This is a very popular activation function in neural networks to introduce non linearity.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
    (10): ReLU()
  )
)
```

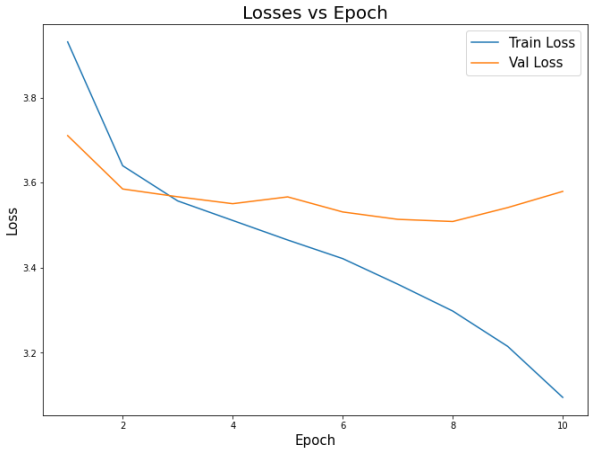
I have added the ReLU activation function layer at the end of the model.

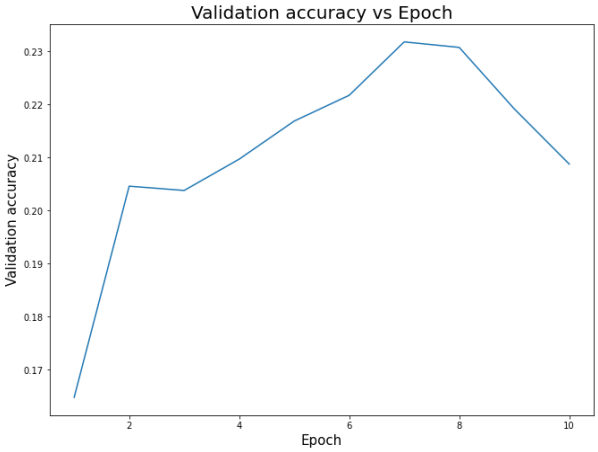
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.930849075317383   Val loss:  3.7105023860931396   Val acc: 0.1648148149251938
Epoch: 2        Train loss: 3.639716148376465   Val loss:  3.5850183963775635   Val acc: 0.20462962985038757
Epoch: 3        Train loss: 3.556999444961548   Val loss:  3.566721200942993    Val acc: 0.20381943881511688
Epoch: 4        Train loss: 3.5110766887664795  Val loss:  3.550445318222046    Val acc: 0.20972223579883575
Epoch: 5        Train loss: 3.465085029602051   Val loss:  3.566462278366089    Val acc: 0.21689815819263458
Epoch: 6        Train loss: 3.421351671218872   Val loss:  3.5310258865356445   Val acc: 0.22175925970077515
Epoch: 7        Train loss: 3.3612897396087646  Val loss:  3.5136752128601074   Val acc: 0.2318287044763565
Epoch: 8        Train loss: 3.298110246658325   Val loss:  3.5085551738739014   Val acc: 0.23078703880310059
Epoch: 9        Train loss: 3.214963912963867   Val loss:  3.5413079261779785   Val acc: 0.21921296417713165
Epoch: 10       Train loss: 3.0950989723205566  Val loss:  3.5793302059173584   Val acc: 0.20879629254341125
```

The minimum validation loss attained is **3.5085** at **8th** epoch. The maximum validation accuracy attained is **23.18%** at **7th** epoch.

Graph Plots





The model seems to overfit the training data from **8th** epoch by looking at the validation accuracy plot.

Comparing different activation functions

LogSigmoid and LogSoftmax seem to be the best activation functions at the end of the network. However, ReLU and Tanh are also good activation functions.

Different pooling strategies

Now, let's look at different pooling strategies which can be used in CNN models.

Max Pooling

This type of pooling takes maximum value in the filter region.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
  )
)
```

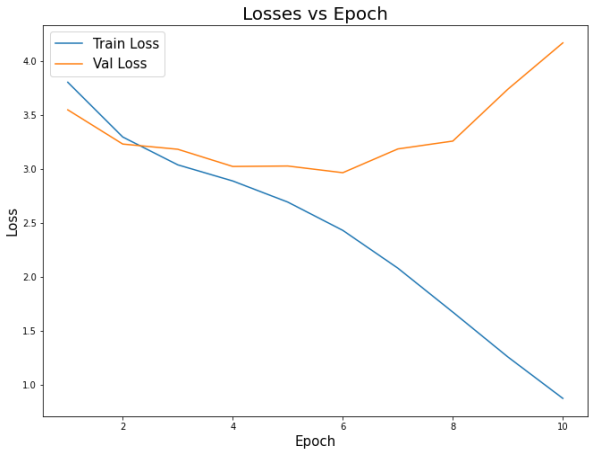
MaxPool layers are added after convolutional layers.

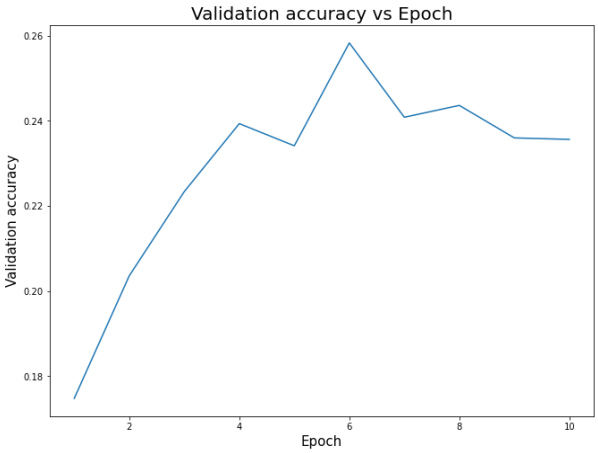
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.8009023666381836  Val loss:  3.5452396869659424   Val acc: 0.17476850748062134
Epoch: 2        Train loss: 3.293173313140869   Val loss:  3.2278521060943604   Val acc: 0.20358796417713165
Epoch: 3        Train loss: 3.0353925228118896  Val loss:  3.179802417755127    Val acc: 0.22337962687015533
Epoch: 4        Train loss: 2.885687828063965   Val loss:  3.02030086517334     Val acc: 0.23935183882713318
Epoch: 5        Train loss: 2.690101385116577   Val loss:  3.0244717597961426   Val acc: 0.23414351046085358
Epoch: 6        Train loss: 2.4279983043670654  Val loss:  2.962477445602417    Val acc: 0.25833332538604736
Epoch: 7        Train loss: 2.0780863761901855  Val loss:  3.1826560497283936   Val acc: 0.24085648357868195
Epoch: 8        Train loss: 1.6704670190811157  Val loss:  3.2556653022766113   Val acc: 0.24363425374031067
Epoch: 9        Train loss: 1.2552039623260498  Val loss:  3.736119270324707    Val acc: 0.2359953671693802
Epoch: 10       Train loss: 0.8708454370498657  Val loss:  4.165708065032959    Val acc: 0.23564815521240234
```

This is the same as that of the baseline model. The minimum validation loss attained is **2.96** at **6th** epoch. The maximum attained validation accuracy is **25.83%** at **6th** epoch.

Graph Plots





We can clearly see that the model is overfitting the training data from the **7th** epoch.

Average Pooling

This type of pooling takes the average value of the filter region values.

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): AvgPool2d(kernel_size=2, stride=2, padding=0)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
  )
)
```

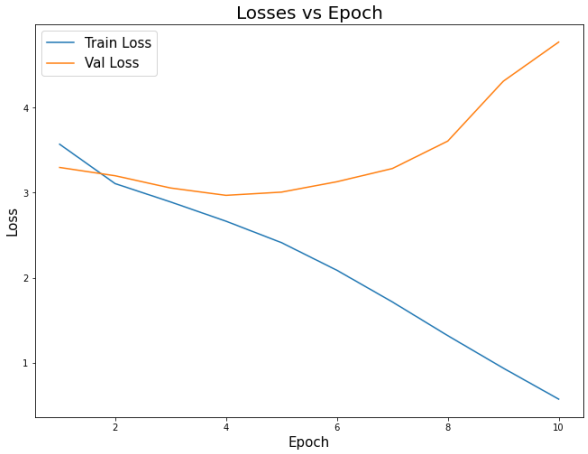
AvgPool layers are added after every convolutional layer.

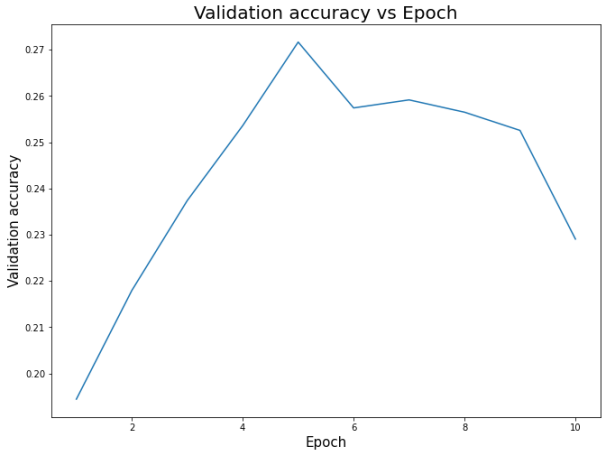
Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.5695390701293945  Val loss:  3.296727180480957    Val acc: 0.1944444328546524
Epoch: 2        Train loss: 3.107388973236084   Val loss:  3.1994874477386475   Val acc: 0.2179398238658905
Epoch: 3        Train loss: 2.892014265060425   Val loss:  3.055555820465088    Val acc: 0.23738425970077515
Epoch: 4        Train loss: 2.6656384468078613  Val loss:  2.969367265701294    Val acc: 0.2535879611968994
Epoch: 5        Train loss: 2.413893461227417   Val loss:  3.007124900817871    Val acc: 0.2716435194015503
Epoch: 6        Train loss: 2.089252471923828   Val loss:  3.129155397415161    Val acc: 0.25740739703178406
Epoch: 7        Train loss: 1.7173936367034912  Val loss:  3.2840628623962402   Val acc: 0.25914350152015686
Epoch: 8        Train loss: 1.3210734128952026  Val loss:  3.6066837310791016   Val acc: 0.25648149847984314
Epoch: 9        Train loss: 0.9398094415664673  Val loss:  4.309243202209473    Val acc: 0.2525462806224823
Epoch: 10       Train loss: 0.5754379034042358  Val loss:  4.769662857055664    Val acc: 0.2290509194135666
```

The minimum validation loss attained is **2.969** at **4th** epoch. The maximum validation accuracy attained is **27.16%** at **5th** epoch.

Graph Plots





We can see that the model starts overfitting from the **6th** epoch by looking at the plots above.

Comparing above pooling techniques

We can see that the plot of losses and validation accuracy is smoother for average pooling than max pooling. And the maximum validation accuracy is more for the model with average pooling than of that with max pooling.

Different optimizers

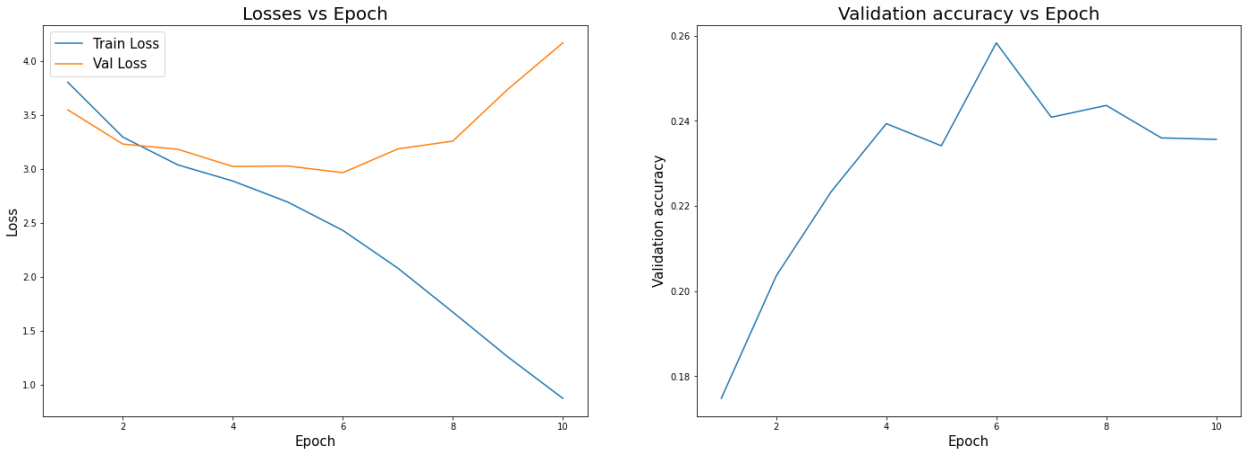
Now, let's look at some optimizers and compare them.

Adam optimizer

Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.8009023666381836  Val loss:  3.5452396869659424   Val acc: 0.17476850748062134
Epoch: 2        Train loss: 3.293173313140869   Val loss:  3.2278521060943604   Val acc: 0.20358796417713165
Epoch: 3        Train loss: 3.0353925228118896  Val loss:  3.179802417755127    Val acc: 0.22337962687015533
Epoch: 4        Train loss: 2.885687828063965   Val loss:  3.02030086517334     Val acc: 0.23935183882713318
Epoch: 5        Train loss: 2.690101385116577   Val loss:  3.0244717597961426   Val acc: 0.23414351046085358
Epoch: 6        Train loss: 2.4279983043670654  Val loss:  2.962477445602417    Val acc: 0.25833332538604736
Epoch: 7        Train loss: 2.0780863761901855  Val loss:  3.1826560497283936   Val acc: 0.24085648357868195
Epoch: 8        Train loss: 1.6704670190811157  Val loss:  3.2556653022766113   Val acc: 0.24363425374031067
Epoch: 9        Train loss: 1.2552039623260498  Val loss:  3.736119270324707    Val acc: 0.2359953671693802
Epoch: 10       Train loss: 0.8708454370498657  Val loss:  4.165708065032959    Val acc: 0.23564815521240234
```

Graph Plots



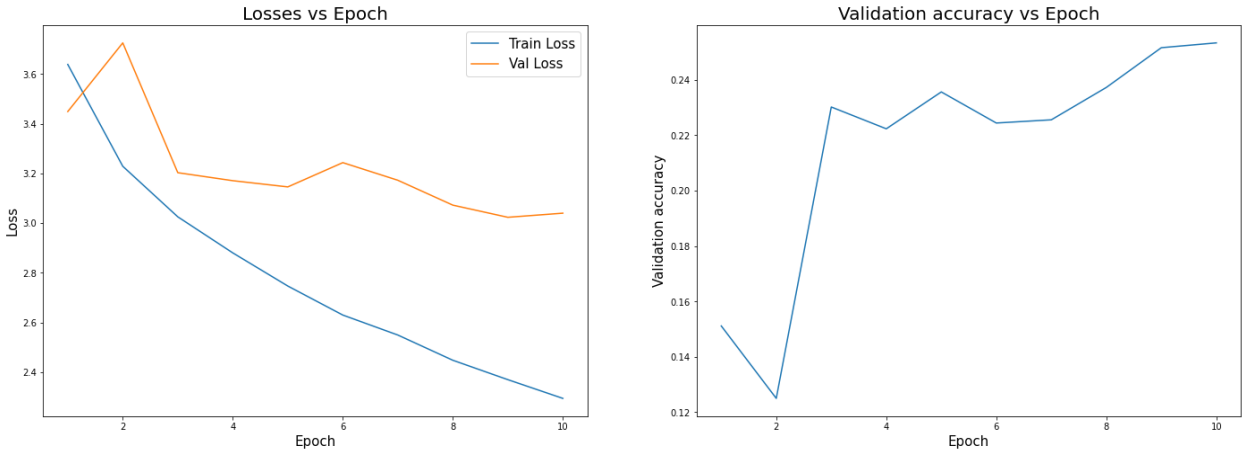
The model starts to overfit the training data from the 7th epoch.

Adagrad optimizer

Loss and Accuracy Values

```
Epoch: 1        Train loss: 3.638272523880005   Val loss:  3.4486865997314453   Val acc: 0.151157408952713
Epoch: 2        Train loss: 3.2285070419311523  Val loss:  3.725010871887207    Val acc: 0.125
Epoch: 3        Train loss: 3.025132894515991   Val loss:  3.2026877403259277   Val acc: 0.23020833730697632
Epoch: 4        Train loss: 2.8799805641174316  Val loss:  3.1705334186553955   Val acc: 0.22233796119689941
Epoch: 5        Train loss: 2.7466585636138916  Val loss:  3.145451307296753    Val acc: 0.23564815521240234
Epoch: 6        Train loss: 2.6298587322235107  Val loss:  3.2431492805480957   Val acc: 0.22442129254341125
Epoch: 7        Train loss: 2.54948091506958    Val loss:  3.1722724437713623   Val acc: 0.22557871043682098
Epoch: 8        Train loss: 2.4479458332061768  Val loss:  3.0721211433410645   Val acc: 0.23726850748062134
Epoch: 9        Train loss: 2.3698978424072266  Val loss:  3.023206949234009    Val acc: 0.2516203820705414
Epoch: 10       Train loss: 2.2945408821105957  Val loss:  3.039752960205078    Val acc: 0.2533564865589142
```

Graph Plots



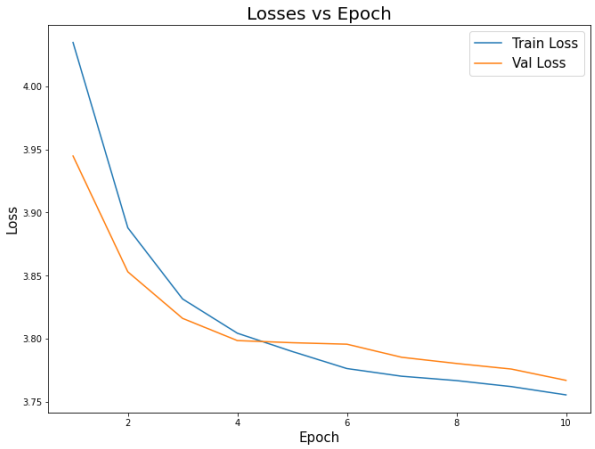
The above plots show that the model is still not overfitting to the data. Hence, the training rate is slow. Due to which the number of epochs required to get a good generalized model is high.

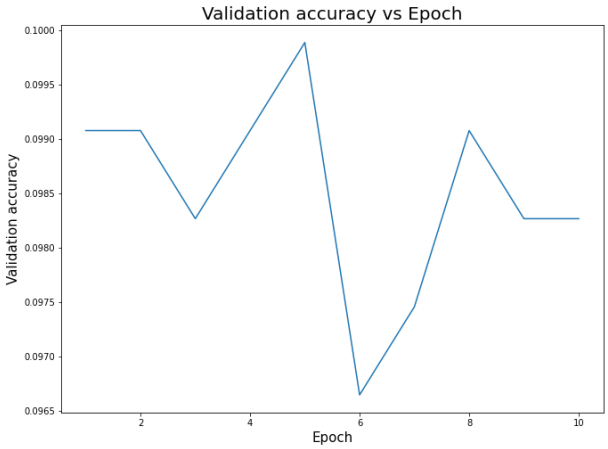
Adadelta optimizer

Loss and Accuracy Values

```
Epoch: 1        Train loss: 4.034656524658203   Val loss:  3.944812297821045    Val acc: 0.09907408058643341
Epoch: 2        Train loss: 3.8879477977752686  Val loss:  3.853069305419922    Val acc: 0.09907408058643341
Epoch: 3        Train loss: 3.8315744400024414  Val loss:  3.8161959648132324   Val acc: 0.09826388210058212
Epoch: 4        Train loss: 3.804443359375      Val loss:  3.7986037731170654   Val acc: 0.09907408058643341
Epoch: 5        Train loss: 3.789989709854126   Val loss:  3.796929121017456    Val acc: 0.09988425672054291
Epoch: 6        Train loss: 3.776458501815796   Val loss:  3.7957451343536377   Val acc: 0.09664352238178253
Epoch: 7        Train loss: 3.7703752517700195  Val loss:  3.7853827476501465   Val acc: 0.09745370596647263
Epoch: 8        Train loss: 3.7669239044189453  Val loss:  3.7804763317108154   Val acc: 0.09907408058643341
Epoch: 9        Train loss: 3.762171745300293   Val loss:  3.7760703563690186   Val acc: 0.09826388210058212
Epoch: 10       Train loss: 3.755606174468994   Val loss:  3.7671077251434326   Val acc: 0.09826388210058212
```

Graph Plots





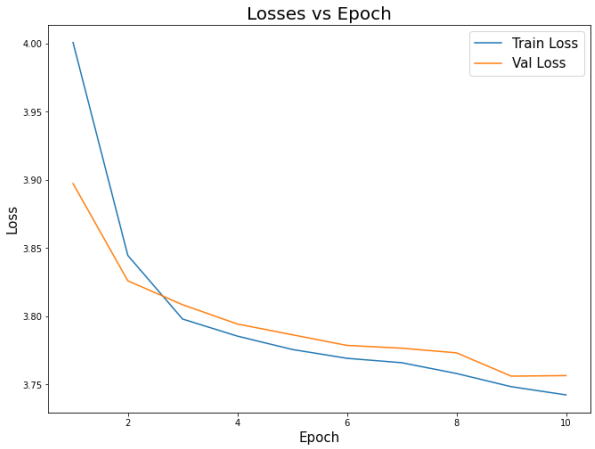
For this model, the plots are a bit inconsistent. The validation loss and training loss is decreasing, but the change is not more. The steps are very small. But when we look at the validation accuracy plot, it is inconsistent. The values are almost the same at all epochs. Hence, the model is not learning properly. Hence, the optimizer is not optimizing steps properly.

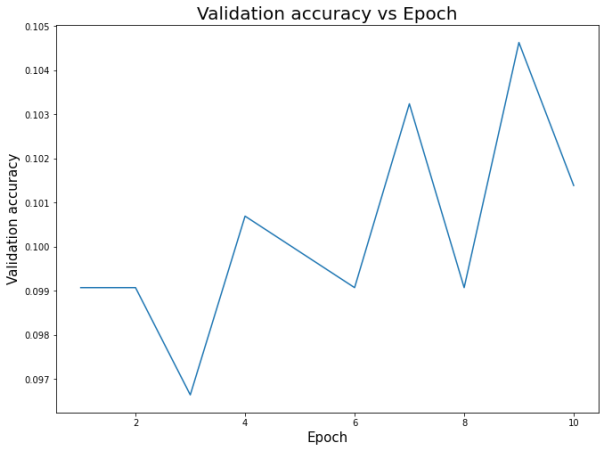
SGD optimizer

Loss and Accuracy Values

```
Epoch: 1        Train loss: 4.000645637512207   Val loss:  3.8972930908203125   Val acc: 0.09907408058643341
Epoch: 2        Train loss: 3.844634771347046   Val loss:  3.826045036315918    Val acc: 0.09907408058643341
Epoch: 3        Train loss: 3.7980148792266846  Val loss:  3.8085105419158936   Val acc: 0.09664352238178253
Epoch: 4        Train loss: 3.7855117321014404  Val loss:  3.79439115524292     Val acc: 0.1006944477558136
Epoch: 5        Train loss: 3.775753974914551   Val loss:  3.7865352630615234   Val acc: 0.09988425672054291
Epoch: 6        Train loss: 3.7692596912384033  Val loss:  3.7787041664123535   Val acc: 0.09907408058643341
Epoch: 7        Train loss: 3.7660157680511475  Val loss:  3.7766549587249756   Val acc: 0.10324074327945709
Epoch: 8        Train loss: 3.7581324577331543  Val loss:  3.773270845413208    Val acc: 0.09907408058643341
Epoch: 9        Train loss: 3.7484548091888428  Val loss:  3.7561895847320557   Val acc: 0.10462962836027145
Epoch: 10       Train loss: 3.742454767227173   Val loss:  3.756591320037842    Val acc: 0.10138888657093048
```

Graph Plots





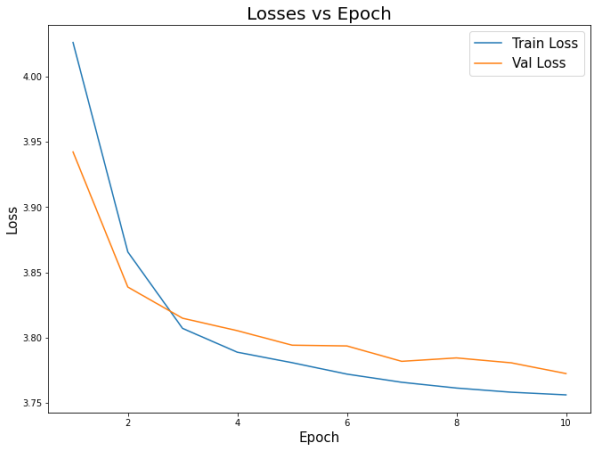
The plots show that the validation loss is decreasing even at 10th epoch. Hence, it is slow in learning. But when we look at the validation accuracy plot, it oscillates a lot, but on average, the accuracy is increasing. This might be because of bigger steps being taken by the optimizer. Maybe by reducing the learning rate further, we can expect to see consistent validation accuracy plots. Hence, this might not be a good optimizer.

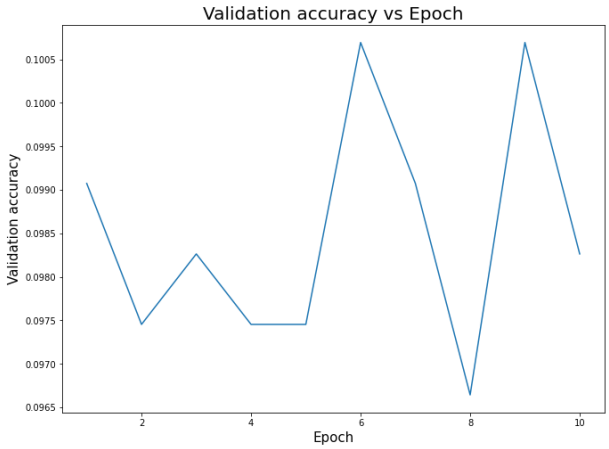
ASGD optimizer

Loss and Accuracy Values

```
Epoch: 1       Train loss: 4.026073455810547   Val loss:  3.9422898292541504   Val acc: 0.09907408058643341
Epoch: 2       Train loss: 3.8657073974609375  Val loss:  3.838827133178711    Val acc: 0.09745370596647263
Epoch: 3       Train loss: 3.8071515560150146  Val loss:  3.814948558807373    Val acc: 0.09826388210058212
Epoch: 4       Train loss: 3.788914203643799   Val loss:  3.8053979873657227   Val acc: 0.09745370596647263
Epoch: 5       Train loss: 3.7808918952941895  Val loss:  3.794296979904175    Val acc: 0.09745370596647263
Epoch: 6       Train loss: 3.772162675857544   Val loss:  3.793710231781006    Val acc: 0.1006944477558136
Epoch: 7       Train loss: 3.7659127712249756  Val loss:  3.781942129135132    Val acc: 0.09907408058643341
Epoch: 8       Train loss: 3.761474609375      Val loss:  3.7845582962036133   Val acc: 0.09664352238178253
Epoch: 9       Train loss: 3.7582972049713135  Val loss:  3.7807915210723877   Val acc: 0.1006944477558136
Epoch: 10      Train loss: 3.756249189376831   Val loss:  3.7725725173950195   Val acc: 0.09826388210058212
```

Graph Plots





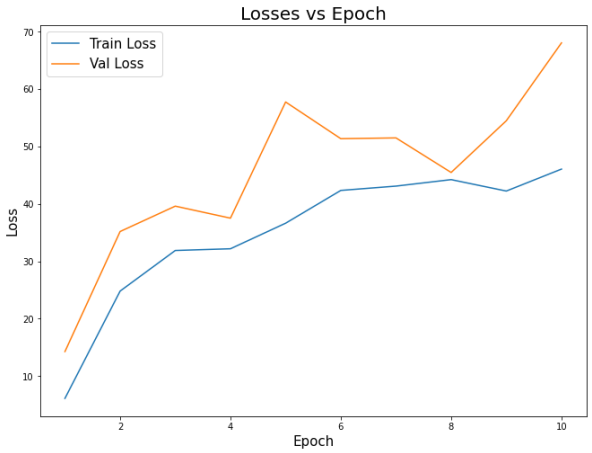
For this model, the plots are a bit inconsistent. The validation loss and training loss is decreasing, but the change is not more. The steps are very small. But when we look at the validation accuracy plot, it is inconsistent. Hence, the model is not learning properly. Hence, the optimizer is not optimizing steps properly.

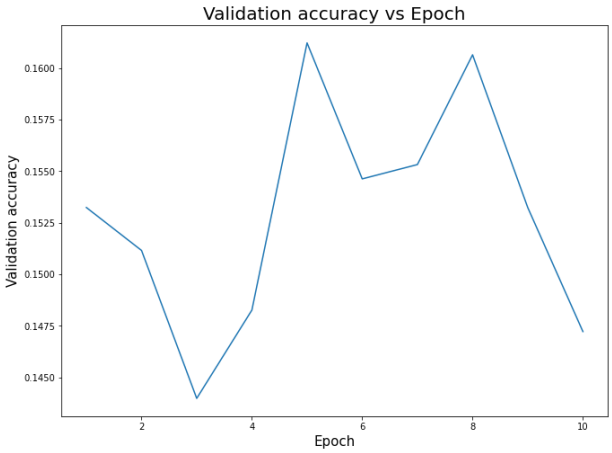
Rprop optimizer

Loss and Accuracy Values

```
Epoch: 1       Train loss: 6.121323108673096   Val loss:  14.27906322479248    Val acc: 0.15324074029922485
Epoch: 2       Train loss: 24.818941116333008  Val loss:  35.20048141479492    Val acc: 0.151157408952713
Epoch: 3       Train loss: 31.895442962646484  Val loss:  39.62272644042969    Val acc: 0.14398148655891418
Epoch: 4       Train loss: 32.19585418701172   Val loss:  37.528717041015625   Val acc: 0.14826388657093048
Epoch: 5       Train loss: 36.64980697631836   Val loss:  57.77468490600586    Val acc: 0.16122683882713318
Epoch: 6       Train loss: 42.362579345703125  Val loss:  51.378746032714844   Val acc: 0.1546296328306198
Epoch: 7       Train loss: 43.11530303955078   Val loss:  51.515357971191406   Val acc: 0.1553240716457367
Epoch: 8       Train loss: 44.24449157714844   Val loss:  45.49114990234375    Val acc: 0.1606481522321701
Epoch: 9       Train loss: 42.25450897216797   Val loss:  54.504520416259766   Val acc: 0.15324074029922485
Epoch: 10      Train loss: 46.080928802490234  Val loss:  68.07528686523438    Val acc: 0.14722222089767456
```

Graph Plots





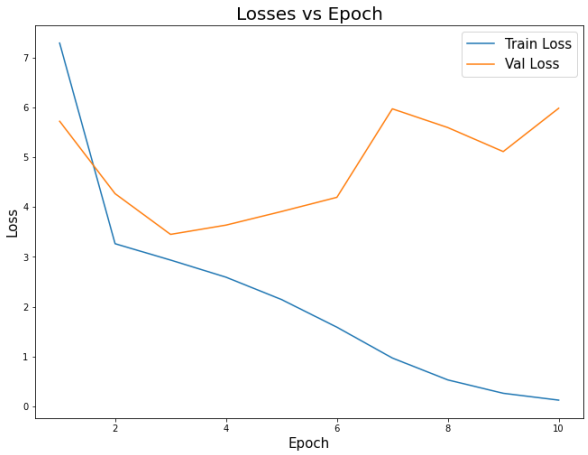
For this model, the plots are a bit inconsistent. The validation loss and training loss is increasing. When we look at the validation accuracy plot, it is inconsistent. The values are almost the same at all epochs. Hence, the model is not learning properly. Hence, the optimizer is not optimizing steps properly. Hence, this is the worst optimizer of all we have seen till now.

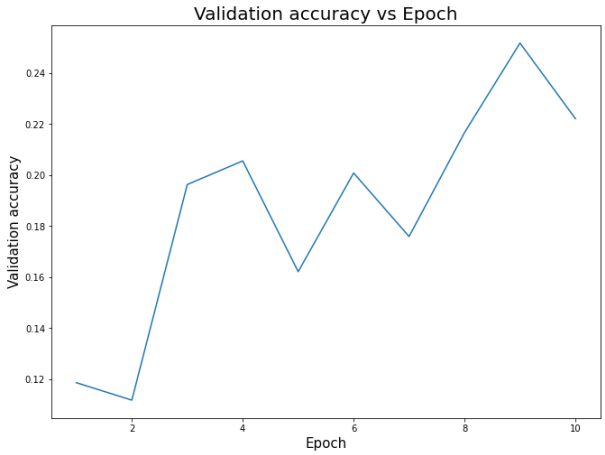
RMSprop optimizer

Loss and Accuracy Values

```
Epoch: 1       Train loss: 7.28810977935791    Val loss:  5.719809055328369    Val acc: 0.11863425374031067
Epoch: 2       Train loss: 3.2629294395446777  Val loss:  4.2671613693237305   Val acc: 0.11180555075407028
Epoch: 3       Train loss: 2.9353654384613037  Val loss:  3.4502570629119873   Val acc: 0.19629628956317902
Epoch: 4       Train loss: 2.591564416885376   Val loss:  3.63614821434021     Val acc: 0.20555557310581207
Epoch: 5       Train loss: 2.1440086364746094  Val loss:  3.9088399410247803   Val acc: 0.16215276718139648
Epoch: 6       Train loss: 1.5886420011520386  Val loss:  4.193541526794434    Val acc: 0.20081017911434174
Epoch: 7       Train loss: 0.970085620880127   Val loss:  5.969281196594238    Val acc: 0.17592592537403107
Epoch: 8       Train loss: 0.5311220288276672  Val loss:  5.593277454376221    Val acc: 0.21666666865348816
Epoch: 9       Train loss: 0.2627912759780884  Val loss:  5.111316204071045    Val acc: 0.2517361044883728
Epoch: 10      Train loss: 0.12533795833587646 Val loss:  5.980954647064209    Val acc: 0.22210648655891418
```

Graph Plots





The validation loss decreases and then increases from 3rd epoch, but the validation accuracy on average seems to be increasing, but not too much. However, the training loss looks steeper initially and then becomes gradual.

Comparing different optimizers

Adam is the best optimizer among all the above seen optimizers. However, RMSprop, Adagrad, SGD can also be used.

Basic augmentation

Data augmentation is very important in learning classifiers. Because they increase the data size. And they make the model more robust to various geometric transformations. Let's look at the results obtained from doing basic data augmentation. Due to GPU usage limit, I've trained the model on only 2000 images from training dataset. And considered following 2 data augmentation techniques,

- Rotate by 90 deg
- Horizontal flipping

Various other data augmentation steps include,

- Adding noise
- Blurring

Model Architecture

```
ClassificationModel(
  (CNN): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=65536, out_features=512, bias=True)
    (8): ReLU()
    (9): Linear(in_features=512, out_features=62, bias=True)
  )
)
```

The model is exactly the same as the baseline model.

Hyperparameters

- **Number of Epochs** = 10
- **Learning rate** = 0.001
- **Batch size** = 64
- **Number of conv layers** = 2
- **Number of fully connected (FC) linear layers** = 2
- **Dropout** = 0

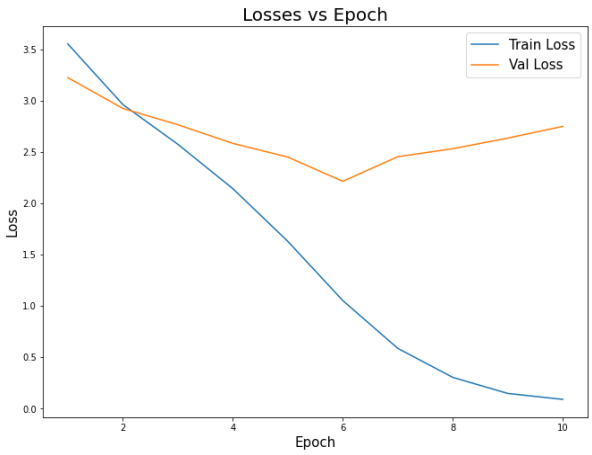
Loss and Accuracy Values

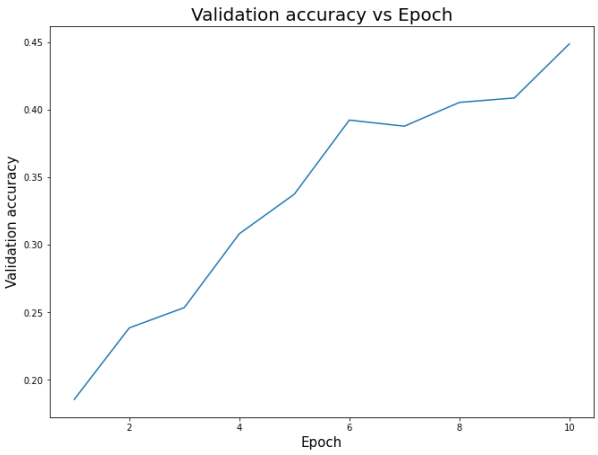
```
Epoch: 1        Train loss: 3.553797483444214   Val loss:  3.225527763366699    Val acc: 0.185546875
Epoch: 2        Train loss: 2.963552474975586   Val loss:  2.9254727363586426   Val acc: 0.23847655951976776
Epoch: 3        Train loss: 2.575881004333496   Val loss:  2.766162395477295    Val acc: 0.2535156309604645
Epoch: 4        Train loss: 2.1436853408813477  Val loss:  2.5844342708587646   Val acc: 0.3082031309604645
Epoch: 5        Train loss: 1.6299006938934326  Val loss:  2.4509477615356445   Val acc: 0.3375000059604645
Epoch: 6        Train loss: 1.0527278184890747  Val loss:  2.214366912841797    Val acc: 0.39238280057907104
Epoch: 7        Train loss: 0.5866565108299255  Val loss:  2.4542980194091797   Val acc: 0.38789063692092896
Epoch: 8        Train loss: 0.3040788769721985  Val loss:  2.532304286956787    Val acc: 0.40546876192092896
Epoch: 9        Train loss: 0.14754484593868256 Val loss:  2.635645866394043    Val acc: 0.4087890684604645
Epoch: 10       Train loss: 0.08977629989385605 Val loss:  2.7500534057617188   Val acc: 0.4488281309604645
```

We can see that the model is training properly on the data. The minimum validation loss attained is 2.214 at 6th epoch. However, the validation loss is not increasing a lot after the 6th epoch. The maximum validation accuracy attained is 44.88% at 10th epoch. Hence, the model has further scope of learning in further epochs. It is because of the data augmentation. As the data size is increased, it takes more time to learn the classifier.

Hence, the training rate is reduced, but it learns a robust model with tradeoff for higher number of epochs required.

Graph Plots





We can see that the validation accuracy is still increasing. Hence, 10 epochs is not sufficient for 2000 images with data augmentation. Hence, the training phase is longer, but learns robust classifiers to geometric transformations to the image.