|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Model | Configuration | Number of Parameters | Training Loss | Training Accuracy | Validation Loss | Validation  Accuracy | Observations | Decision |
|  | CNN+LSTM  (Model 2) | Epochs = 10,  Feature map = [8, 16, 32, 64, 128]  **Dense = [1000,500,5]**  Layer = 4  Batch size = 10  Epochs = 10  Learning rate patience = 5 | 3,974,609 | 1.6063 | 0.2754 | 1.6405 | 0.1917 | We see the accuracy is poor and loss is high. Hence the model is overfitting. | We add batch normalization Layers after convolutions. |
|  | CNN+LSTM  (Model 3) | Epochs = 10,  Feature map = [8, 16, 32, 64, 128]  Dense = [1000,500,5]  **Add Batch Normalizations**  Batch size = 10  Epochs = 20  Learning rate patience = 5 | 3,974,705 | 1.6027 | 0.2657 | 1.4743 | 0.3750 | Slight improvement but the model is still not performing well. | We will reduce the dense layer neurons to original 256,128 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | CNN+LSTM  (Model 1) | Epochs = 10,  Feature map = [8, 16, 32, 64, 128]  **Dense = [256,128,5]**  Layer = 4  Batch size = 10  Epochs = 10  Learning rate patience = 5 | 982,613 | 1.2777 | 0.4541 | 1.1535 | 0.5700 | Model Stopped Learning after a few iterations. There was no improvement in loss. | We will try to increase the learning rate. Actually reduce the learning patience. |
|  | CNN+LSTM | Epochs = 10,  Feature map = [8, 16, 32, 64, 128]  Dense = [256,128,5]  Layer = 4  Batch size = 10  Epochs = 10  **Learning rate patience = 2** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s.NO. | Model | Configuration | Result | Decision+Explanation |
|  | CNN(Conv2D)+RNN | Use LSTM ,  epochs = 10,  feature\_map = [8, 16, 32, 64, 128]  dense = [128, 256, 5]  layer = 4  batchsize = 10  epochs = 10 | Each epoch is taking over 20 mins to train on google colab. | Too high training time. Try smaller batch size |
| 2 | CNN(Conv2D)+RNN | Use LSTM ,  epochs = 10,  feature\_map = [8, 16, 32, 64, 128]  dense = [1000,500,5]  layer = 4  batchsize = 10  epochs = 10 | Each epoch is taking over 20 mins to train on google colab. | Too high training time. Try smaller batch size |
| 3. | CNN(Conv2D)+RNN | Use LSTM ,  epochs = 10,  feature\_map = [8, 16, 32, 64, 128]  layer = 4  batchsize = 10  Add 2 batch normalization layers  dense\_layer\_size = [1000,500,5]  epochs = 20 | loss: 1.2854 - categorical\_accuracy: 0.4686 - val\_loss: 1.4341 - val\_categorical\_accuracy: 0.2833 | increase batch size |
| 4. | CNN(Conv2D)+RNN | Use LSTM ,  epochs = 10,  feature\_map = [8, 16, 32, 64, 128]  layer = 4  batchsize = 100  Add 2 batch normalization layers  dense\_layer\_size = [1000,500,5]  epochs = 20 | loss: 0.3958 - categorical\_accuracy: 0.8537 - val\_loss: 4.9297 - val\_categorical\_accuracy: 0.2100 | Model is overfitting |
| 5 |  | Use LSTM, epochs = 10, ReduceLROnPlateau( monitor="val\_loss", factor=0.1, patience=10, verbose=0, mode="auto", min\_delta=0.0001, cooldown=0, min\_lr=0, \*\*kwargs) |  |  |
| 6 |  | LSTM  bach\_size 663 |  |  |
| 7 |  | image size 84\*84 |  |  |
|  |  |  |  |  |
|  |  | Different batch normalization |  |  |
|  |  | No augmentation |  |  |
|  |  | optimiser change |  |  |
| 4. | CNN(Conv2D)+RNN | GRU  filters = [8, 16, 32, 64, 128, 256]  dense = [1000,500,5]  kernel = (3, 3)  layers = 5  batch\_size = 10  num\_epochs = 10  optimiser Adam 0.001 |  |  |
| 5. | CNN(Conv2D)+RNN | Remove dropouts from 2 and 3rd convolution layer  GRU  filters = [8, 16, 32, 64, 128, 256]  dense = [1000,500,5]  kernel = (3, 3)  layers = 5  batch\_size = 10  num\_epochs = 20  optimiser Adam 0.001 | Cat\_loss came out to nan |  |
| 6. | CNN(Conv2D)+RNN |  |  |  |
| 7. | CNN(Conv2D)+RNN | Add affine transformation to data, 5 epochs, 100 videos only. | loss: 0.8562  categorical\_accuracy: 0.7267  val\_loss: 3.1973  val\_categorical\_accuracy: 0.4800 |  |
| 8. | CNN(Conv2D)+RNN | Add affine transformation to data,  100 videos only ,  10 epochs  Size = 120 | loss: 0.7928  categorical\_accuracy: 0.7267  val\_loss: 3.3824  val\_categorical\_accuracy: 0.4500 |  |
|  |  | Add affine transformation to data,  10 epocs,  100 videos.  Size reduced to 84 | Model is still over fitting but  loss: 0.9580 - categorical\_accuracy: 0.6967 - val\_loss: 2.6072 - val\_categorical\_accuracy: 0.4200 |  |
|  |  | Image Size = 84\*84  GRU  filters = [8, 16, 32, 64, 128, 256]  dense = [1000,500,5]  kernel = (3, 3)  layers = 5  batch\_size = 10  num\_epochs = 20  optimiser Adam 0.001 | loss: 0.7743  categorical\_accuracy: 0.7250  val\_loss: 1.8595  val\_categorical\_accuracy: 0.3500 |  |
|  |  | Epochs = 10  Videos = 663  Size= 84\*84  Batch size = 30 | Model is drastically overfitting  loss: 1.7697  categorical\_accuracy: 0.4396  val\_loss: 2.0745  val\_categorical\_accuracy: 0.2000 |  |
|  |  | Epochs = 10  Videos = 663  Size= 120\*120  Batch size = 30 | loss: 1.6568  categorical\_accuracy: 0.4300  val\_loss: 2.0831  val\_categorical\_accuracy: 0.5250 |  |
|  |  | Dense layer = 64, 32  Epochs = 10  Videos = 100  Size= 120\*120  Batch size = 10 | loss: 0.8811  categorical\_accuracy: 0.8300  val\_loss: 5.1926  val\_categorical\_accuracy: 0.3700 |  |
|  |  | Videos = 663  Size = 120\*120  Epochs = 10  Batch size = 30 | loss: 2.3570  categorical\_accuracy: 0.3478  val\_loss: 6.6927  val\_categorical\_accuracy: 0.2500 |  |
|  | CNN(Conv2D) + RNN | GRU  filters = [8, 16, 32, 64, 128, 256]  dense = [1000, 500, 250, 5]  kernel = (3, 3)  layers = 5  batch\_size = 10  num\_epochs = 20  optimiser Adam 0.001 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | change dense layers = 500,250,5 | Best- loss = loss: 0.9867 - categorical\_accuracy: 0.6425 - val\_loss: 2.0180 - val\_categorical\_accuracy: 0.3083  Best accuracy = loss: 1.1444 - categorical\_accuracy: 0.5700 - val\_loss: 1.2195 - val\_categorical\_accuracy: 0.5167 |  |
|  |  | dense\_layer\_size = [1000,500,5] | loss: 1.1989 - categorical\_accuracy: 0.5169 - val\_loss: 1.3898 - val\_categorical\_accuracy: 0.3750 |  |
|  |  | Add 2 batch normalization layers  dense\_layer\_size = [500,250,5] | best loss =loss: 0.9688 - categorical\_accuracy: 0.6667 - val\_loss: 0.9403 - val\_categorical\_accuracy: 0.5667  best validation |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | Add 2 batch normalization layers  dense\_layer\_size = [1000,500,5] | loss: 1.0714 - categorical\_accuracy: 0.5990 - val\_loss: 0.9876 - val\_categorical\_accuracy: 0.6417 |  |
|  |  | Add 2 batch normalization layers  dense\_layer\_size = [1000,500,5]  epochs = 20 | loss: 0.9868 - categorical\_accuracy: 0.6329 - val\_loss: 0.9841 - val\_categorical\_accuracy: 0.6333 |  |
|  |  |  |  |  |
|  | Conv3D | filters = [8,16,32,64]  dense = [256, 128, 5]  kernel = (3, 3, 3)  layers = 4  batch\_size = 10  num\_epochs = 10  optimiser SGD |  | A good accuracy for starting, Let’s try increasing number of epoch sand changing optimiser |
|  | Conv3D | filters = [8,16,32,64]  dense = [256, 128, 5]  kernel = (3, 3, 3)  layers = 4  batch\_size = 10  num\_epochs = 20  optimiser Adam |  | Decent accuracy, less validation loss. Let’s try adding more perceptrons in the dense layer. |
|  | Conv3D | filters = [8,16,32,64]  dense = [1000, 500, 5]  kernel = (3, 3, 3)  layers = 4  batch\_size = 10  num\_epochs = 20  optimiser Adam |  | Accuracy has improved significantly. Let’s try adding one more layer with dropout |
|  | Conv3D | filters = [8, 16, 32, 64, 128]  dense = [1000, 500, 5]  kernel = (3, 3, 3)  layers = 5  batch\_size = 10  num\_epochs = 20  optimiser Adam |  | Accuracy has not increased much. Let’s add another layer with Batch Normalization |
|  | Conv3D | filters = [8, 16, 32, 64, 128, 256]  dense = [1000, 500, 5]  kernel = (3, 3, 3)  layers = 6  batch\_size = 10  num\_epochs = 20  optimiser Adam |  | Accuracy has not increased much. |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Conv3D  4 layer, 3,3,3 kernel  optimiser Adam  epoch - 20  batch\_size 10 |  |  |  |
| Conv3D  dense = [1000, 500, 5]  4 layer, 1,3,3 kernel  optimiser Adam  epoch - 20  batch\_size 10  Batch normalisation in first 3 layers and dropout in 4th |  | loss: 0.3612 - categorical\_accuracy: 0.8557 - val\_loss: 0.5799 - val\_categorical\_accuracy: 0.7800 |  |
| Conv3D  dense = [1000, 500, 5]  5 layer, 1,3,3 kernel  optimiser Adam  epoch - 20  batch\_size 10  Batch normalisation in first 3 layers and dropout in 4th and 5th |  | loss: 0.4773 - categorical\_accuracy: 0.8159 - val\_loss: 0.5750 - val\_categorical\_accuracy: 0.7700 |  |

Image size

Batch