Please be advised to view the model architectures within the provided ipynb file while reviewing this report to gain a better understanding on the summarised tabular information.

# **Experiment: Varying Size of Each Batch**

*Hypothesis*: Smaller batch sizes will result in better learning for the dataset.

*Constants*: Epochs = 25, Optimiser – SGD(lr=0.01, momentum=0.5), Max pooling = (3,3,3), Sequence length = 30, Img size = (120,120,3), Conv Layers = 2, FC Layers = 1, Hidden Activation = ReLU

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | | Observations |
| Model 1 | 3D Conv | Batch size = 63 | 0.7511 | 0.7094 | 0.39 | 1.3745 | 0.28 | The model has overfit significantly. There is very little learning occurring. | |
| Model 2 | 3D Conv | Batch size = 7 | 0.8371 | 0.4562 | 0.7500 | 0.7605 | 8.3 | Significantly less overfitting | |

***Conclusion***: The model with the smaller batch size performed significantly better than that with the larger size. However, the best accuracy and loss for the validation set did not occur at the final epoch with the final epoch seeing val accuracy of 57% and a loss of 1.1640. These results indicate that batch sizes have significant impact on the training of models.

# **Experiment: Varying Sequence Length**

***Hypothesis***: The different between each subsequent frames is small leading to overfitting so by only providing every other frame, we can reduce overfitting.

***Constants***: Epochs = 25, Optimiser = SGD(lr=0.01, momentum=0.5), filter = (3,3,3), Max pooling = (3,3,3), Batch size = 7, Img size = (120,120,3), Conv Layers = 2, FC Layers = 1, Hidden Activation = ReLU

Note, Model 2, essentially evaluates the same but using a sequence length of 30, therefore only one model with seq length of 15 needs to be built.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 3 | 3D Conv | Sequence Length = 15 | 0.9713 | 0.1051 | 0.82 | 0.5512 | 2.7 | Model overfits but training time is improved. Performance similar to model with 30 for sequence length. |

***Conclusion***: Reducing the length of each input sequence halved the time taken to train the model and did not negatively impact performance vs the full sequence model. However, overfitting still occurs.

# **Experiment: Varying Image Size**

***Hypothesis***: The input images come in varying sizes of 360x360 and 120x160, resizing all images to 120x120 results in loss of information. If an intermediary value is used of 160, we should retain more information and performance.

***Constants***: Epochs = 25, Optimiser = SGD(lr=0.01, momentum=0.5), filter = (3,3,3), Max pooling = (3,3,3), Batch size = 7, Sequence length = 15, Conv Layers = 2, FC Layers = 1, Hidden Activation = ReLU

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 4 | 3D Conv | Img size = (160,160,3) | 0.9759 | 0.1114 | 0.77 | 0.6167 | 4.7 | Model overfits more on training and performs worse on validation |

***Conclusion***: Image sizes of 160x160 on this dataset seem to promote the overfitting of the model on the training set. The result is worse loss values on the validation set. It seems having a lower resolution input when there are varied resolution data may be the best option. With the lowest resolution matching the lowest dimension of an input image, in this case 120.

# **Experiment: Varying Conv Depth**

***Hypothesis***: A deeper network should be capable of learning more features and thus be capable of giving better performance or at least the same as a shallower layer.

***Constants***: Epochs = 25, Optimiser = SGD(lr=0.01, momentum=0.5), filter = (3,3,3), Batch size = 7, Sequence length = 15, Img size = (160,160,3), FC Layers = 1, Hidden Activation = ReLU

Note – to increase depth, max pooling pool size needed to be changed.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 5 | 3D Conv | Conv layers = 3  Max pooling = 2,2,2 | 0.8235 | 0.5275 | 0.64 | 0.7843 | 7.4 | Model overfit less on training but still the validation performance was lower than previous models |

***Conclusion***: Increasing depth and decreasing pool size resulted in higher loss values on both validation and training sets. However, the impact on one vs the other on the model will need to be teased out in a model shallower model with pool size being the only parameter changed from previous runs.

# **Experiment: Varying Pool Size**

***Hypothesis***: To Increase Depth, the pool size needed to be dropped to 2,2,2. This was a change of 2 parameters, depth and pool size. In order to understand the impact of pool size vs depth, a model should be run as a base line to gauge how pool size affects training.

***Constants***: Epochs = 25, Optimiser = SGD(lr=0.01, momentum=0.5), filter = (3,3,3), Batch size = 7, Sequence length = 15, Img size = (160,160,3), Conv layers = 2, FC Layers = 1, Hidden Activation = ReLU

Note – in order to increase depth, max pooling pool size needed to be changed.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 6 | 3D Conv | Max pooling = 2,2,2 | 0.9970 | 0.036 | 0.69 | 1.082 | 44.2 | Extreme overfitting, nearly perfectly learns training set. |

***Conclusion***: Significant rise in the number of parameters due to the smaller pool size. Model overfits worse than all others. It appears that at least for this model, a deeper model with a pool size of 2,2,2 performs well just not as well as a shallower model with pool size 3,3,3.

# **Experiment: Varying Hidden Layer Activation Function**

***Hypothesis***: Leaky ReLU can have better performance is deeper networks, those with lots of noise etc. The down sampled pictures are noisier than their original counterpart therefore, leaky ReLU should help to lessen overfitting and improve overall model performance.

***Constants***: Epochs = 25, Optimiser = SGD(lr=0.01, momentum=0.5), filter = (3,3,3), Max pooling = (3,3,3), Batch size = 7, Sequence length = 15, Conv Layers = 2, FC Layers = 1,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 7 | 3D Conv | Hidden layer activation = LeakyReLU | 0.9759 | 0.0935 | 0.76 | 0.6462 | 2.7 | Overfits but the learning is more continuous with a steady rise in acc and drop in loss |

***Conclusion***: This is a shallow network and therefore the choice of LeakyReLU may not have been advisable, however, the model did perform well, almost as good as the Model 3 on which it was based. LeakyReLU in shallower networks can give similar performance to ReLU. It should be noted as well that unlike model 3 whose performance plateaued, this model showed continued decline in loss and increase in average accuracy after each successive epoch. It is possible with more training time, this model could have performed better than model 3.

# **Experiment: ConvRNN Models - GRU vs LSTM**

***Hypothesis***: LSTM have 4 times as many parameters than basic RNN and GRUs have 3 times as many. However, both models should perform similarly with the GRU model being faster to train.

***Constants***: Epochs = 25, Optimiser = SGD(lr=0.01, momentum=0.5), filter = (3,3), Max pooling = (3,3), Batch size = 7, Sequence length = 15, Conv Layers = 2, RNN cells = 32 , Hidden Layer activation = ReLU

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 8 | ConvRNN-LSTM | LSTM cells | 0.9925 | 0.1166 | 0.74 | 0.7829 | 0.7 | Rapidly overfit, initial rapid gains for validation accuracy |
| Model 9 | ConvRNN - GRU | GRU cells | 0.9879 | 0.1055 | 0.74 | 0.6391 | 0.5 | Similar training time to LSTM. Lower loss value |

***Conclusion***: The training times were similar but as expected GRU had fewer parameters and performed just as well as the LSTM model. In fact, the GRU model attained better loss values with fewer parameters. For this dataset at least, a GRU is the better option to proceed with. Both option though overfit quite a bit.

# **Experiment: ConvRNN Models - Adam vs SGD Optimiser**

***Hypothesis***: Adam, adaptive moment estimation, is an optimisation algorithm that can update learning rates for different parameters so in theory it should provide a better route to convergence than basic SDG with momentum.

***Constants***: Epochs = 25, filter = (3,3), Max pooling = (3,3), Batch size = 7, Sequence length = 15, Conv Layers = 2, RNN cells = 32 , Hidden Layer activation = ReLU

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 10 | ConvRNN-GRU | Optimiser – Adam() | 0.8824 | 0.4072 | 0.73 | 0.7774 | 0.5 | The model overfit much less and decreases in loss more continuous |

***Conclusion***: Adam has allowed for the model to more easily learn the general pattern of the data as the train accuracies did not reach 99-100% as in previous models with SGD(). While the loss values were higher, the model was also a bit slower to train so it is possible that with more training epochs, this model could have achieved 80+ accuracies for the validation and training sets.

# **Experiment: ConvRNN Models – Lower Batch Size**

***Hypothesis***: A smaller batch size should allow for better learning of the RNN model

***Constants***: Epochs = 25, Optimiser = Adam(), filter = (3,3), Max pooling = (3,3), Batch size = 5, Sequence length = 15, Conv Layers = 2, RNN cells = 32 , Hidden Layer activation = ReLU

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 11 | ConvRNN-GRU | Batch size = 5 | 0.8356 | 0.5215 | 0.71 | 0.8317 | 0.5 | Overfitting reduced more |

***Conclusion***: Batch size plays an important role in whether a model with overfit or not. In fact, this model only began overfitting in the final 15 epochs vs the final 20 in other RNN models. Even though it over fit the difference between the best training accuracy and validation accuracy was smaller than other RNN models.

# **Experiment: ConvRNN Models – More Cells**

***Hypothesis***: More GRU cells should enable better modeling of the temporal aspects of the data.

***Constants***: Epochs = 25, Optimiser = Adam(), filter = (3,3), Max pooling = (3,3), Batch size = 5, Sequence length = 15, Conv Layers = 2, Hidden Layer activation = ReLU

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 12 | ConvRNN-GRU | GRU cells = 64 | 0.8989 | 0.3877 | 0.71 | 0.7177 | 2.7 | Overfitting but less significantly than other changes. Worse than fewer cells |

***Conclusion***: Increasing the number of cells without making changes elsewhere in the network promoted more overfitting.

# **Experiment: ConvRNN Models – Pool Size**

***Hypothesis***: Smaller pool sizes should preserve more detail

***Constants***: Epochs = 25, Optimiser = Adam(), filter = (3,3), Batch size = 5, Sequence length = 15, Conv Layers = 2, RNN Cells = 32, Hidden Layer activation = ReLU

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 13 | ConvRNN-GRU | Pool size = (2,2) stride =2 | 0.6833 | 0.9193 | 0.53 | 1.0854 | 2.7 | Stable model, training and val loss decreased with each other. |

***Conclusion***: Best loss for validation occurred with a training loss of 0.9771 and accuracy of 0.6667. This model was much more stable than before indicating that the stride and pool size when maxpooling can have a very big impact on your performance especially when finer grained features need to be compared and highlighted.

# **Experiment: ConvRNN Models – Deeper**

***Hypothesis***: With more convolutional layers more spatial information can be learned improving performance

***Constants***: Epochs = 25, Optimiser = Adam(), filter = (3,3), Batch size = 5, Sequence length = 15 , RNN Cells = 32, Hidden Layer activation = ReLU, Pool size = (2,2) stride =2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameter | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 14 | ConvRNN-GRU | Conv Layers = 3 | 0.6305 | 0.9867 | 0.62 | 1.1091 | 1.4 | Very stable, accuracy around 60% for val and train |

***Conclusion***: The model may be better able to identify spatial and some temporal differences but lacks the ability to properly separate and classify.

# **Experiment: ConvRNN Models – Deeper II**

***Hypothesis***: With more convolutional layers and skip connections more spatial information can be learned improving performance

***Constants***: Epochs = 25, Optimiser = Adam(), filter = (3,3), Batch size = 5, Sequence length = 15 , Conv Layers = 3, Hidden Layer activation = ReLU, Pool size = (2,2) stride =2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameters | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 15 | ConvRNN-GRU |  |  |  |  |  |  |  |

***Conclusion***: The model may be better able to identify spatial and some temporal differences but lacks the ability to properly separate and classify reaching accuracies in the 80s. This is the best performing model with overfitting being a smaller issue. Perhaps with more data, better results could be achieved.

# **Experiment: Transfer Learning – Resnet Untrainable**

***Hypothesis***: Resnet was the first model to use residual/skip connections. Knowing this, the model should be capable of non random performance as it has been trained to identify features in images.

***Constants***: Epochs = 40, Optimiser = Adam(), Resnet Layers = untrainable

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameters | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 15 | ConvRNN-GRU | none | 0.6802 | 0.8049 | 0.64 | 0.82 | 23.9 | No overfitting. The model reached stable accuracies of 60-65% |

***Conclusion***: The model did not overfit on the training set but got stuck for training and validation accuracy around 60-65%

# **Experiment: Transfer Learning – Resnet Trainable**

***Hypothesis***: A comparison of the default Resnet weights to newly learned rates by allowing backpropagation to update all weights in the model

***Constants***: Epochs = 40, Optimiser = Adam()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameters | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 15 | ConvRNN-GRU | Resnet.layers=trainable | 0.2398 | 1.6091 | 0.3 | 1.5783 | 23.9 | Model does not learn, changing weights gave horrible results |

***Conclusion***: The model failed to learn with accuracies stuck below 50%. Resnet does not need to have all weights updated in order to be used for transfer learning, the final layers are likely all that need to be optimised for a new dataset.

# **Experiment: Transfer Learning – Resnet 50/50**

***Hypothesis***: Comparison model with only the final 50% of layers trainable

***Constants***: Epochs = 40, Optimiser = Adam()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | Model Type | Varied Parameters | Train  Acc. | Best Train  Loss | Val  Acc. | Best Val  Loss | No. Params in Million | Observations |
| Model 15 | ConvRNN-GRU | Resnet.layers – final 50% layers trainable | 1.0 | 0.0211 | 0.86 | 0.3633 | 23.9 | Model quickly learns but then begins to overfit |

***Conclusion***: The model performed better than both previous models BUT it began to overfit which indicates that fewer layers in resnet need to be trainable. This clearly shows the power of transfer learning as the basic resnet model likely only needs the very final layers updated in order to give good results on new images classification datasets.