Experiments – Gesture Recognition Assignment

Following table summarizes the experiments carried out. The models are represented as sequential layers just like we specify in our python code.

i. The **output layer** is not specified in the model column, it is to be considered as `'Dense(5, activation='softmax')'`
ii. The **activation function** used for any layer that requires it is `'relu'`.
iii. The Conv2D and Pooling2D layers are TimeDistributed.

Input shape: (20, 120, 120, 3)
Number of epochs: 30

No.	Model	Result	Decision + Explanation
1	Conv3D(32, kernel = (3,3,3)) MaxPooling3D(pool = (2,2,2)) Conv3D(64, kernel = (3,3,3)) MaxPooling3D(pool = (2,2,2)) Conv3D(128, kernel = (3,3,3)) MaxPooling3D(pool = (2,2,2)) Flatten() Dense(128) -- optimiser: SGD(learning_rate=0.05) batch_size: 64	This model is underfitting and is not able to learn the patterns in the data **loss**: 1.0598 **categorical_accuracy**: 0.5867 **val_loss**: 1.4216 **val_categorical_accuracy**: 0.3500	Let's try to solve the underfitting by changing the model topology to some extent. We can try to first apply convolution and pooling on individual images, rather than the entire sequence. This can be done using kernels of the shape – (1, 3, 3)
2	Conv3D(32, kernel = (1,3,3)) MaxPooling3D(pool = (1,2,2)) Conv3D(64, kernel = (1,3,3)) MaxPooling3D(pool = (1,2,2)) Conv3D(128, kernel = (1,3,3)) MaxPooling3D(pool = (1,2,2)) Conv3D(32, kernel = (3,3,3)) MaxPooling3D(pool = (2,2,2)) Conv3D(64, kernel = (3,3,3)) MaxPooling3D(pool = (2,2,2)) Flatten() Dense(128) -- optimiser: SGD(learning_rate=0.05) batch_size: 64	This model is performing even worse than model 1. The model is performing even worse than a random classifier. **loss**: 1.5898 **categorical_accuracy**: 0.2353 **val_loss**: 1.6013 **val_categorical_accuracy**: 0.2500	Let's try to invert our approach and first apply convolution and pooling on our sequence and convert the sequence into a single image. Then we can apply pooling on the single image. This can be done by first using kernels of the shape – (3, 1, 1)

<table>
<tr><th></th><th></th><th></th><th></th></tr>
<tr><td>3</td><td>Conv3D(32, kernel = (2,1,1))
MaxPooling3D(kernel = (2,1,1))

Conv3D(64, kernel = (2,1,1))
MaxPooling3D(kernel = (2,1,1))

Conv3D(128, kernel = (1,3,3))
MaxPooling3D(kernel = (1,2,2))

Conv3D(32, kernel = (1,3,3))
MaxPooling3D(kernel = (2,2,2))

Conv3D(64, kernel = (1,3,3))
MaxPooling3D(kernel = (1,2,2))

Conv3D(128, kernel = (1,3,3))
MaxPooling3D(kernel = (1,2,2))

Flatten()
Dense(128)
----
optimiser: SGD(learning_rate=0.05)
batch_size: 64</td><td>This model is still performing quite poor, and underfitting the data.

loss: 1.2340
categorical_accuracy: 0.5415
val_loss: 1.4350
val_categorical_accuracy: 0.3600</td><td>It seems like the models with Conv3D are not able to identify the patterns in our data.

Therefore, now would be a good time to move to a CNN + RNN (GRU) model</td></tr>
<tr><td>4</td><td>Conv2D(32, kernel = (3,3))
MaxPooling2D(kernel = (2,2))
Dropout(0.25)

Conv2D(32, kernel = (3,3))
MaxPooling2D(kernel = (2,2))
Dropout(0.25)

Conv2D(32, kernel = (3,3))
MaxPooling2D(kernel = (2,2))
Dropout(0.25)

Flatten()
GRU(32)
Dropout(0.25)
----
optimiser: SGD(learning_rate=0.05)
batch_size: 64</td><td>This model is performing very poor. It is underfitting to a great extent, and worse than a random classifier

loss: 1.6100
categorical_accuracy: 0.2021
val_loss: 1.6019
val_categorical_accuracy: 0.1900</td><td>It looks like the model structure is not well suited to learn the patterns in the data. The data is too complicated for this model which is very simple.

We can increase the neurons in the convolution layers and also add another GRU for better interpretation of the sequences.</td></tr>
<tr><td>5</td><td>Conv2D(32, kernel = (3,3))
MaxPooling2D(kernel = (2,2))
Dropout(0.25)

Conv2D(64, kernel = (3,3))
MaxPooling2D(kernel = (2,2))
Dropout(0.25)</td><td>This model has performed well compared to model 4. It can identify the patterns in the data to some extent.
Adding more neurons and an additional GRU layer has worked well.</td><td>Although this model has a good accuracy score for train as well as validation, we can see that the model has overfit to some extent.</td></tr>
</table>

	Conv2D(128, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Flatten() GRU(128) Dropout(0.25) GRU(128) Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 64	**loss**: 1.0100 **categorical_accuracy**: 0.7074 **val_loss**: 1.1922 **val_categorical_accuracy**: 0.6100	And although the accuracy scores are not high enough to consider this model as efficient, we can experiment further. Let's try to solve some of the overfitting by trying to Reduce Learning Rate on Plateau of validation loss.
6	Conv2D(32, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Conv2D(64, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Conv2D(128, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Flatten() GRU(128) Dropout(0.25) GRU(128) Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 64 callback: LRReduceOnPlateau(factor=0.5)	With the call back, the model is not performing as well as model 5. **loss**: 1.6651 **categorical_accuracy**: 0.5354 **val_loss**: 1.7998 **val_categorical_accuracy**: 0.4500	The reduced learning rate is slowing the model learning. Let's try to use a different optimizer, without any callbacks.
7	Conv2D(32, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Conv2D(64, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Conv2D(128, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25)	The model performance is very poor with Adam optimizer. The model is worse than a random classifier. **loss**: 1.7870 **categorical_accuracy**: 0.1916 **val_loss**: 1.7848 **val_categorical_accuracy**: 0.2300	Adam optimizer is not suited for our model. It is causing too much underfitting. One option for the next experiment would be to use RMSProp optimizer, but this is like Adam and doesn't seem like a very lucrative option. So rather, let's try to fine tune model 5 by removing the

	Flatten() GRU(128) Dropout(0.25) GRU(128) Dropout(0.25) -- optimiser: Adam(learning_rate=0.05) batch_size: 64		dropouts added to the GRU layers.
8	Conv2D(32, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Conv2D(64, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Conv2D(128, kernel = (3,3)) MaxPooling2D(kernel = (2,2)) Dropout(0.25) Flatten() GRU(128) GRU(128) -- optimiser: SGD(learning_rate=0.05) batch_size: 64	There doesn't seem much improvement over model 5. **loss**: 0.9334 **categorical_accuracy**: 0.7179 **val_loss**: 1.3443 **val_categorical_accuracy**: 0.5200	The convolutional (CNN) part of the model seems unable to learn the patterns in the data. It would be a good option now to move to transfer learning and replace the CNN part with a pre trained model. We can use the keras model – **MobileNet**
9	MobileNet(weights='imagenet') GlobalAveragePooling2D() Flatten() GRU(128) Dropout(0.25) GRU(128) Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 64	This model took too long to train and does not yield any significant difference over model 5. **loss**: 0.8334 **categorical_accuracy**: 0.6172 **val_loss**: 2.2443 **val_categorical_accuracy**: 0.2201	**MobileNet** has a lot of parameters due to which the performance is slow. This is an issue for us as we have limited computing resources. Let's use **MobileNetV2** which is significantly lighter and has lesser number of paraments
10	MobileNetV2(weights='imagenet') GlobalAveragePooling2D() Flatten() GRU(128) Dropout(0.25) GRU(128)	The training throws the following error – `Resource exhausted: OOM when allocating tensor with shape[1920,144,30,30] and type float`	The size of the tensors used in MobileNetV2 are too huge to be fit in available memory. Let's reduce the batch size to 32.

	Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 64		
11	MobileNetV2(weights='imagenet') GlobalAveragePooling2D() Flatten() GRU(128) Dropout(0.25) GRU(128) Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 32	The model has performed better than any of the models that we trained previously. **loss**: 0.2204 **categorical_accuracy**: 1.0000 **val_loss**: 1.0171 **val_categorical_accuracy**: 0.7200	Although, as we can see, the model has overfit a lot. The training accuracy is 1 meaning the model has completely learned the training data. Let's reduce the complexity of our model by removing one GRU layer.
12	MobileNetV2(weights='imagenet') Dropout(0.3) GlobalAveragePooling2D() Flatten() GRU(128) Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 32	The model has less overfitting as compared to model 11. **loss**: 0.2505 **categorical_accuracy**: 1.0000 **val_loss**: 0.8192 **val_categorical_accuracy**: 0.8300	Reducing the model complexity has helped reduce the overfitting. Let's try to reduce overfitting further by adding a dropout layer for our MobileNetV2.
13	MobileNetV2(weights='imagenet') Dropout(0.3) GlobalAveragePooling2D() Flatten() GRU(128) Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 32	The model has less overfitting as compared to model 12. **loss**: 0.2039 **categorical_accuracy**: 1.0000 **val_loss**: 0.5373 **val_categorical_accuracy**: 0.8600	The dropout layer has helped reducing the overfitting by a minute amount. We can still try to reduce overfitting further by Reducing Learning Rate on Plateau.
14	MobileNetV2(weights='imagenet') Dropout(0.3) GlobalAveragePooling2D() Flatten() GRU(128) Dropout(0.25) -- optimiser: SGD(learning_rate=0.05) batch_size: 32	The validation accuracy has dropped significantly. This is not a good model. **loss**: 0.4151 **categorical_accuracy**: 1.0000 **val_loss**: 0.9864 **val_categorical_accuracy**: 0.7300	Reduce LR on Plateau is interfering with our model's learning causing it to overfit wildly. Let's try to build upon model 13, and reduce overfitting by introducing a kernel_regularizer='l1' to our output layer.

	callback: LRReduceOnPlateau(factor=0.5)		
15	**Final Model:** MobileNetV2(weights='imagenet') Dropout(0.3) GlobalAveragePooling2D() Flatten() GRU(128) Dropout(0.25) Dense(activation='softmax', kernel_regularizer='l1') --- optimiser: SGD(learning_rate=0.05) batch_size: 32	This model is performing very well. There is some over fitting here as w ell. **loss**: 0.2024 **categorical_accuracy**: 1.0000 **val_loss**: 0.5048 **val_categorical_accuracy**: 0.8900	This is the best model that we can squeeze out of our dataset. If we have more data, we can further increase the validation accuracy to match that of the training accuracy.