Preface:

Models are too large to upload to black board and thus kept local. By running the CNN_GPU python notebook, it can download the required data. The inceptionV4 notebook can download the label used for imagenet, and it need the CNN_GPU to download the tiny imagenet for the data. I'll try to include the tiny imagenet dataset but I don't know if blackboard is going to like it.

The video is kept on youtube, accessible via this link: https://youtu.be/AiNXWjw9gPE

CNN

For inception net, I'm using V3 instead of V4. V3 is more mature and has better support on tensorFlow, will V4 is only available with timm package on hugging face. Thus I also included V3 for comparison

All models: VGG16, VGG19, Resnet50, Resnet100 and InceptionV3 are included from the tensorFlow models. The version of tensorFlow I'm using is 2.10.0 because it stopped supporting GPU after that version on Windows Native. All three networks are appended with a flatten then dense maxpool layer with softmax activation function, and 200 nodes as output. All but the output layers are hold to be untrainable, loaded with pre-trained weights ("DEFAULT" is trained on imageNet), then finetuned to work with tiny imagenet dataset. The accuracy is determined by the top 1 category (instead of "within the top 5"). Which explains why the accuracy is generally not high. I finetuned VGG19, resnet50 and inceptionV3 for 50 epochs, and VGG16 and Resnet100 for 15.

Before I figured out GPU setup with tensorflow, I used CPU to train 10epochs for VGG19, ResNET50 and InceptionV3. The finetuning training time is reported in the table below.

During 38th epoch for InceptionV3, my GPU run out of resource. I'm going to use the stats from 37th epoch for that model. Because it's something I added in as additional comparison, I'm not too worried about its results. However, it running out of resource, and a quick analysis of the accuracy for finetuning made me thing I might not want to (or have the capability and time to) finetune InceptionV4.

Model	VGG16	VGG19	ResNET50	ResNET100	InceptionV3
Trainable Parameters Count	5,017,800	5,017,800	409,800	409,800	409,800
Toal Parameters Count	19,732,488	25,042,184	23,974,600	43,036,360	22,212,584
GPU train time (1080ti, per epoch)	440s	435s	430s	435s	775s
CPU train time (AMD Ryzen 9 '7900X, per epoch)	N/A	1150s	1570s	N/A	835s

Here are the accuracy results on the validation set for each model. Note: for InceptionV3, I'm using the 37th epoch training accuracy. For InceptionV4, I forgo finetuning based on results from other models, and directly applied that model to the tiny imagenet's validation set.

Model	VGG16 (15epochs)	VGG19	Resnet50	Resnet100 (15epochs)	InceptionV3 (37^{th} epoch)	InceptionV4 (no finetuning)
Accuracy	0.304	0.3064	0.1128	0.0356	0.0938	0.498
Loss	89.47	62.85	9.14	154.10	4.65	N/A

It seems like larger models such as resnet50, resnet100, inception net needs more epochs to finetune the model and provide a good result: For the same number of epochs trained, VGG19 has significantly better accuracy than resnet50, and same relationship holds between VGG16 and resnet100 or inceptionV3. Between the same family of models, VGG16 reached similar accuracy only after 15 epochs, comparing VGG19's 50. For resnet family, resnet100 after 15epochs has significantly lower accuracy than resnet50. For inceptionV3, after 50 epochs, it reaches similar accuracy compared to that of Resnet50.

The reason behind this performance difference between families of model is likely due to how simple that family of models are. VGG16 and 19 are much simpler than resnets or inception nets. While VGG is only convolution layers followed by pooling layers, resnet use residual blocks for deeper network, inception net even more complex. The simplicity of the model makes the model easier to achieve better results with limited fine-tuning training time.

Within each family, the smaller the network, the easier it is to fine tune and better the result is with limited epochs. VGG16 only needed 15 epochs to reach the similar accuracy of VGG19. Looking at resnet50's 15th epoch (accuracy 0.1136, loss 13.3814), it's also way ahead of resnet100's 15th epoch. This is due to the smaller size of the network, makes it less complex and thus easier to finetune.

For inceptionV4, after looking at the low accuracy result from inceptionV3 (and the fact I don't have nearly enough time and resource to finetune it), I directly checked it on the validation set. Reaching an accuracy of 0.498, the highest amongst the bunch, it seems like sometimes unfinished fine-tuning will yield worse results than just applying the model to the dataset. This is because when we finetune the model from 1000 classes to 200 classes, we are required to train a dense layer at the output, which consists 5,017,800 nodes for VGG or 409,800 nodes for ResNet/Inception to train. From further research on people's result published on the internet, it seems that if I have enough resources, finetuning will eventually yields a better accuracy result than no finetuning.

RNN

Result table:

Data length	LSTM (mse)	GRU (mse)	Bidirectional (mse)	DeepRNN (mse)
Baseline	5.44	4.96	7.35	9.05
2*Baseline (716)	4.50	3.30	3.71	2.19
4*Baseline (1446)	1.84	1.07	0.61	0.07
8*Baseline (2906)	0.21	0.054	0.038	0.018
Data length	LSTM (time)	GRU (time)	Bidirectional (time)	DeepRNN (time)
Baseline	14.01	14.48	15.75	13.67
2*Baseline (716)	18.5	21.3	22.27	18.77
4*Baseline (1446)	30.3	36.11	36.36	32.88
8*Baseline (2906)	53.88	61.9	67.06	56.5

The test is run on a time series dataset, with the same “base” data and increasing in size by doubling its length each time. The “new” data is formed by simple appending the current data behind itself. The “accuracy” of prediction is gauged by the MSE value, and the speed of the algorithm is judged by its execution time.

Overall, we can see that there’s a trade off between efficiency of training and performance. The more complex the RNN structure, it might take longer to train and has better performance than the simple/baseline LSTM.

As expected, DeepRNN and Bidirectional RNN should be better at more complex datasets (longer ones). From the results, we see that when the length of the data increases, DeepRNN and bidirectional RNN both drastically increases their performance. When the length of the data is at the longest, DeepRNN has the best MSE of 0.018, way better than LSTM’s 0.21.

As for training time, because Bidirectional is the most complex as it need the reverse of the sequency as well, it takes the longest time to train. When dataset gets larger, that time difference between LSTM and Bidirectional becomes more and more noticeable.