

Richard Andreas

CS542 Class Challenge Report

a) Describe the architectures used in detail: layers, layer dimensions, dropout layers, etc. for both tasks. List the optimizer, loss function, parameters, and any regularization used in both tasks

Task 1 (Binary Classification)

True
Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
vgg16 (Functional)	(None, 7, 7, 512)	14714688
batch_normalization (Batch Normalization)	(None, 7, 7, 512)	2048
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257
activation (Activation)	(None, 1)	0
=====		
Total params: 21,139,777		
Trainable params: 6,424,065		
Non-trainable params: 14,715,712		

I decided to use the pretrained VGG16 Model on the ImageNet dataset for transfer learning. After the convolution layers of VGG16, I applied Batch Normalization and a Flatten Layer, Fully Connected Layer and Dropout Layer. Activations for the first and last fully Connected Layers are ReLU and sigmoid respectively.

The probability of the dropout layer is 0.3 and the number of units in the first connected layer is 256 and the last layer is 1 (for binary classification)

I applied several regularization techniques, Batch Normalization and Dropout, and L2 Kernel Regularization in the first fully connected Layer.

For compilation, I used the Adam Optimizer with a learning rate of 0.001, with the binary cross entropy loss function. The model is trained over 40 epochs with a batch size of 10, 10 steps per epoch for training and 2 steps per epoch for validation.

Task 2 (Categorical Classification):

Model: "sequential"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
batch_normalization_94 (Batch Normalization)	(None, 5, 5, 2048)	8192
flatten (Flatten)	(None, 51200)	0
densefn (Dense)	(None, 256)	13107456
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 4)	1028
activation_94 (Activation)	(None, 4)	0
Total params: 34,919,460		
Trainable params: 13,112,580		
Non-trainable params: 21,806,880		

I decided to use the pretrained InceptionV3 Model on the ImageNet dataset for transfer learning, freezing the first 300 layers and training the rest. After the convolution layers of VGG16, I applied Batch Normalization and a Flatten Layer, Fully Connected Layer and Dropout Layer. Activations for the first and last fully Connected Layers are ReLU and softmax respectively.

The probability of the dropout layer is 0.3 and the number of units in the first connected layer is 256 and the last layer is 4 (for categorical classification)

I applied several regularization techniques, Batch Normalization and Dropout.

For compilation, I used the Adam Optimizer with a learning rate of 0.001, with the categorical cross entropy loss function. The model is trained over 100 epochs with a batch size of 10, 10 steps per epoch for training and 5 steps per epoch for validation.

b) Comparison of the performance of different architectures for the second task and relating this to the architecture and parameter settings used.

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
batch_normalization (Batch Normalization)	(None, 7, 7, 512)	2048
flatten (Flatten)	(None, 25088)	0
densefn (Dense)	(None, 256)	6422784
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 4)	1028
activation (Activation)	(None, 4)	0
Total params: 21,140,548		
Trainable params: 6,424,836		
Non-trainable params: 14,715,712		

Model: "sequential"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
batch_normalization_94 (Batch Normalization)	(None, 5, 5, 2048)	8192
flatten (Flatten)	(None, 51200)	0
densefn (Dense)	(None, 256)	13107456
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 4)	1028
activation_94 (Activation)	(None, 4)	0
Total params: 34,919,460		
Trainable params: 13,112,580		
Non-trainable params: 21,806,880		

I have used 2 different pre trained models (VGG16 and InceptionV3) for task2 and compared the various results between both.

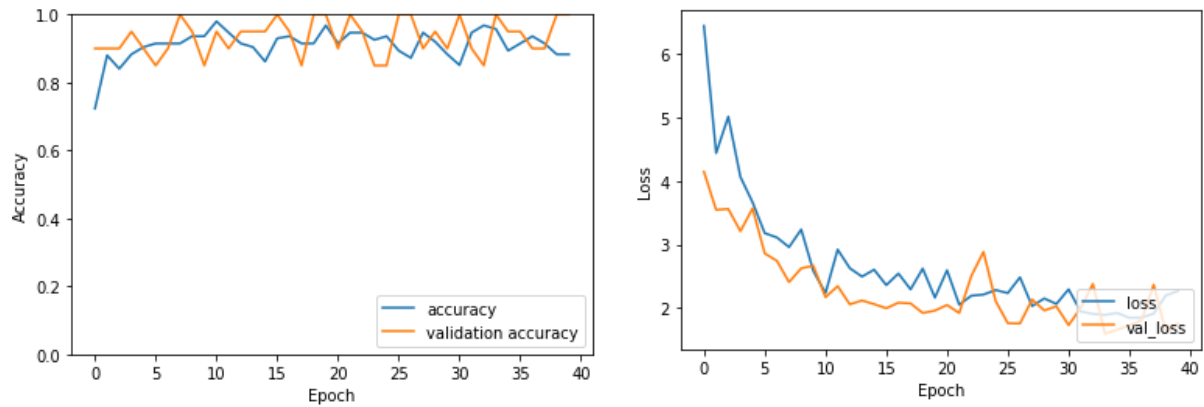
	VGG16	InceptionV3
Time taken per Epoch (seconds)	20-25	12 - 18
Time per Step	2-3 s/step	0.8-1s/step
Time taken	2359 seconds	1060 seconds
Total Parameters	21,140,548	34,919,460
Test Loss	1.2161	1.57811
Test Accuracy	0.5833	0.75

I used the same parameters settings between both architectures to get a fair comparison between both models for training and validation: Adam Optimizer, learning rate of 0.001, 100 epochs, 10 steps per epoch, batch size of 10, 5 steps per epoch for validation.

As you can see the time taken for VGG16 to run is approx. 2 times longer than that of InceptionV3, even though there are less parameters. In terms of results, the test loss of VGG16 (1.2161) is lower than InceptionV3 (1.578), but the accuracy of VGG16 (0.5833) is worse than that of InceptionV3 (0.75)

c) Plot and comment on the accuracy and the loss for both tasks

Task1:

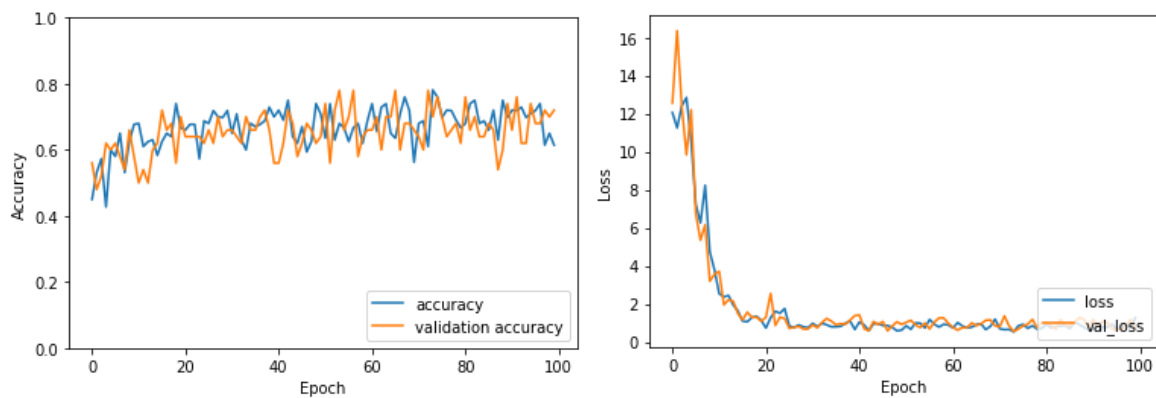


Test Loss: 1.7654

Test Accuracy: 1.0

The accuracy and validation accuracy for task 1 fluctuates between 0.85 and 1.0. The validation accuracy hits 1.0 several times while fitting the model. The loss and validation do fluctuate, however there is a decreasing gradient and starts to plateau around the 25th epoch to around 1.0.

Task 2:



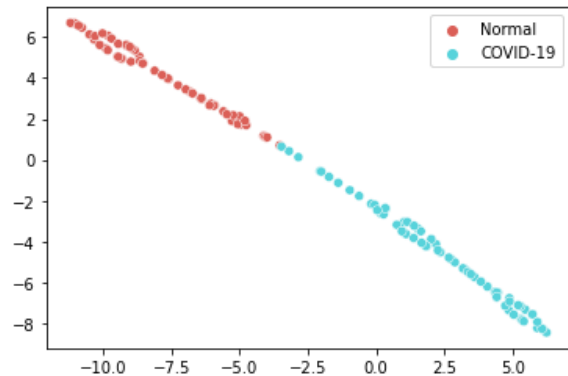
Test loss: 0.7389

Test accuracy: 0.75

The accuracy and validation accuracy for task 2 fluctuates between 0.5 and 0.7 after 20 epochs. For the loss and validation, there is a decreasing gradient and starts to plateau around the 30th epoch to around 1.0 from 12.

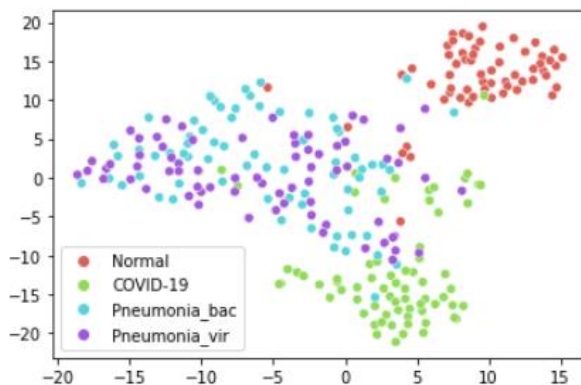
d) Plot and comment on the t-SNE visualizations

Task 1:



From the t-SNE plot of task 1, the COVID-19 and Normal Features are clearly separated into 2 different clusters. Meaning the Neural Network can differentiate the unique features between COVID-19 and Normal X-Rays.

Task 2:



However, when looking at the t-SNE plot of task 2, the model is able to differentiate between COVID-19, Normal and Pneumonia X-rays, but unable to differentiate between the 2 variants of Pneumonia (Bacterial and Viral). The top right (red) are mostly Normal Xray's, while bottom (yellow) is mainly COVID-19. However, the middle cluster is mixed with both Pneumonia bacterial and Pneumonia Viral, even though they are both in one cluster, a better model would have been able to separate these 2 variations. So, from this plot, my model is able to separate them into 3 distinct clusters instead of 4.

e) Bonus: Run the training on a GPU on the SCC cluster and include a CPU vs. GPU training time comparison by taking snapshots from your terminal

Jupyter Notebook

This app will launch a Jupyter Notebook server on a compute node.

List of modules to load (space separated)

Select Modules

Pre-Launch Command (optional)

Interface



Working Directory

Select Directory

The directory to start Jupyter in. (Defaults to home directory.)

Extra Jupyter Arguments (optional)

Number of hours

Number of cores

Number of gpus

GPU compute capability



Project



Extra qsub options

☒ I would like to receive an email when the session starts

For Task1:

Running on local CPU

→ localhost:8888/notebooks/task1_template.ipynb

jupyter task1_template Last Checkpoint: 21 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

```
x = train_batches,
epochs=NUM_EPOCHS,
validation_data= valid_batches,
steps_per_epoch = STEP_SIZE_TRAIN,
batch_size=BATCH_SIZE,
validation_steps = STEP_SIZE_VALID
)

end = time.time()
10/10 [=====] - 16s 2s/step - loss: 1.7893 - accuracy: 0.9690 - val_loss:
1.5917 - val_accuracy: 1.0000
Epoch 35/40
10/10 [=====] - 16s 2s/step - loss: 1.9637 - accuracy: 0.8682 - val_loss:
1.6573 - val_accuracy: 0.9500
Epoch 36/40
10/10 [=====] - 16s 2s/step - loss: 1.7845 - accuracy: 0.9442 - val_loss:
1.7199 - val_accuracy: 0.9500
Epoch 37/40
10/10 [=====] - 16s 2s/step - loss: 1.8672 - accuracy: 0.9439 - val_loss:
1.8049 - val_accuracy: 0.9000
Epoch 38/40
10/10 [=====] - 16s 2s/step - loss: 2.0210 - accuracy: 0.9002 - val_loss:
2.3667 - val_accuracy: 0.9000
Epoch 39/40
10/10 [=====] - 16s 2s/step - loss: 2.2281 - accuracy: 0.9226 - val_loss:
1.6307 - val_accuracy: 1.0000
Epoch 40/40
10/10 [=====] - 16s 2s/step - loss: 2.1520 - accuracy: 0.9146 - val_loss:
1.7534 - val_accuracy: 1.0000

In [8]: print("Model RunTime (seconds):",end - start)
Model RunTime (seconds): 748.4690515995026
```

Running on SCC using 1 GPU

scc-ondemand1.bu.edu/node/scc-204/29660/notebooks/task1_template.ipynb

jupyter task1_template Last Checkpoint: 19 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

[20 points] Binary Classification: COVID-19 vs. Normal

```
In [1]: import os

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from tensorflow.keras import regularizers
import seaborn as sns
import time

# os.environ['OMP_NUM_THREADS'] = '1'
# os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
tf.__version__
```

```
Out[1]: '2.4.1'
```

```
In [2]: tf.config.list_physical_devices()
```

```
Out[2]: [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
In [3]: tf.test.gpu_device_name()
```

```
Out[3]: '/device:GPU:0'
```

scc-ondemand1.bu.edu/node/scc-204/29660/notebooks/task1_template.ipynb

jupyter task1_template Last Checkpoint: 20 minutes ago (unsaved changes)

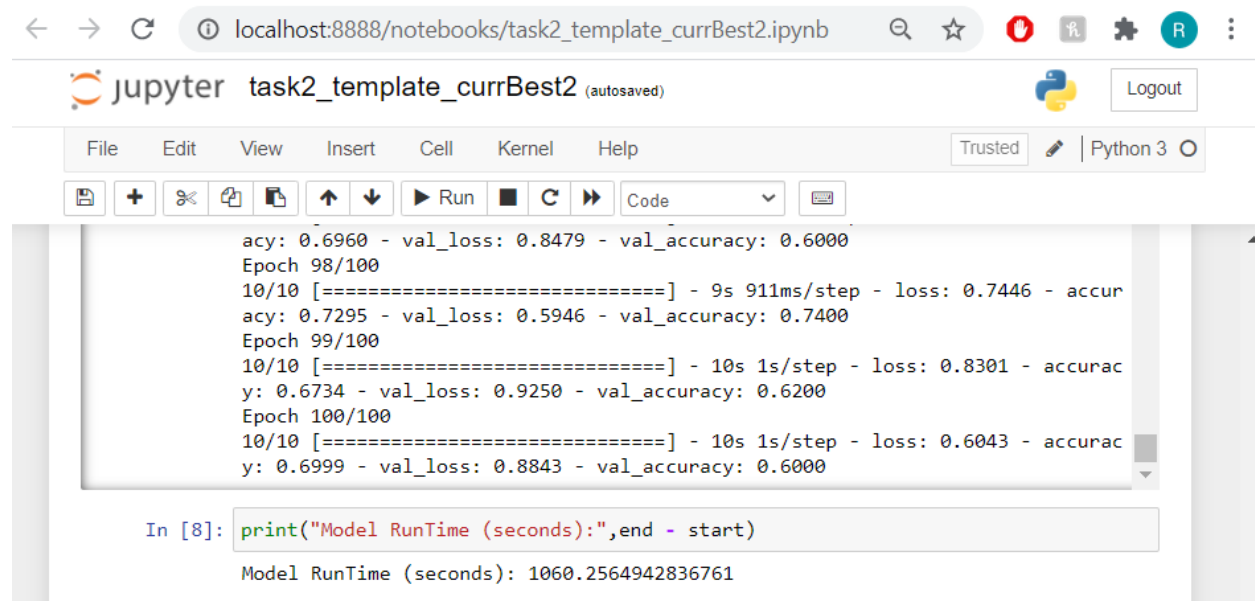
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
Epoch 31/40
10/10 [=====] - 3s 275ms/step - loss: 0.2357 - accuracy: 0.9357 - val_loss: 0.0525 - val_accuracy: 0.9500
Epoch 32/40
10/10 [=====] - 3s 323ms/step - loss: 0.3068 - accuracy: 0.9248 - val_loss: 0.8298 - val_accuracy: 0.9000
Epoch 33/40
10/10 [=====] - 3s 280ms/step - loss: 0.1484 - accuracy: 0.9669 - val_loss: 0.3221 - val_accuracy: 0.9500
Epoch 34/40
10/10 [=====] - 3s 295ms/step - loss: 0.2382 - accuracy: 0.9690 - val_loss: 0.0409 - val_accuracy: 0.9500
Epoch 35/40
10/10 [=====] - 3s 301ms/step - loss: 0.1759 - accuracy: 0.9472 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 36/40
10/10 [=====] - 3s 284ms/step - loss: 0.2879 - accuracy: 0.9357 - val_loss: 0.5885 - val_accuracy: 0.9000
Epoch 37/40
10/10 [=====] - 3s 273ms/step - loss: 0.1365 - accuracy: 0.9416 - val_loss: 0.0000 - val_accuracy: 1.0000
```

```
In [10]: print("Model RunTime (seconds):",end - start)

Model RunTime (seconds): 127.04223704338074
```


Task2: Running on Local CPU

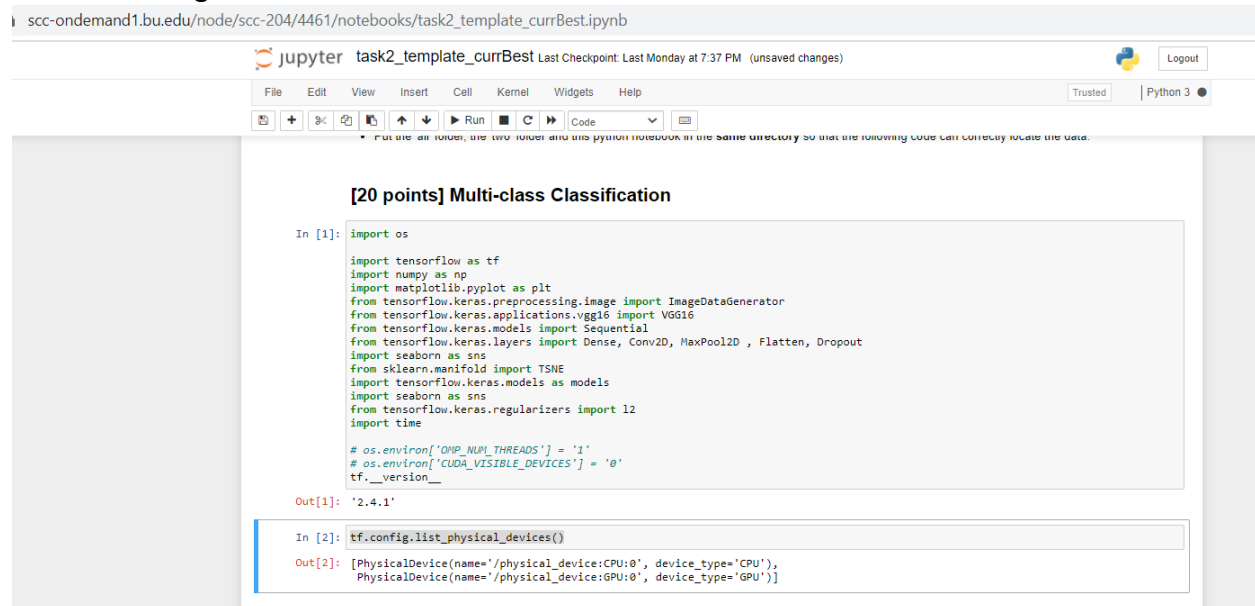


```
acy: 0.6960 - val_loss: 0.8479 - val_accuracy: 0.6000
Epoch 98/100
10/10 [=====] - 9s 911ms/step - loss: 0.7446 - accuracy: 0.7295 - val_loss: 0.5946 - val_accuracy: 0.7400
Epoch 99/100
10/10 [=====] - 10s 1s/step - loss: 0.8301 - accuracy: 0.6734 - val_loss: 0.9250 - val_accuracy: 0.6200
Epoch 100/100
10/10 [=====] - 10s 1s/step - loss: 0.6043 - accuracy: 0.6999 - val_loss: 0.8843 - val_accuracy: 0.6000

In [8]: print("Model RunTime (seconds):",end - start)

Model RunTime (seconds): 1060.2564942836761
```

Task2: Running on GPU on the SCC



```
[20 points] Multi-class Classification

In [1]: import os

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
import seaborn as sns
from sklearn.manifold import TSNE
import tensorflow.keras.models as models
import seaborn as sns
from tensorflow.keras.regularizers import l2
import time

# os.environ['OMP_NUM_THREADS'] = '1'
# os.environ['CUDA_VISIBLE_DEVICES'] = '0'
tf.__version__

Out[1]: '2.4.1'

In [2]: tf.config.list_physical_devices()

Out[2]: [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
In [8]: start = time.time()

history = model.fit(
    x = train_batches,
    epochs=100,
    validation_data= valid_batches,
    steps_per_epoch = 10,
    batch_size=10,
    validation_steps = STEP_SIZE_VALID
)

end = time.time()

Epoch 94/100
10/10 [=====] - 3s 279ms/step - loss: 0.6320 - accuracy: 0.7132 - val_loss: 0.9150 - val_accuracy: 0.6400
Epoch 95/100
10/10 [=====] - 3s 322ms/step - loss: 1.0306 - accuracy: 0.6614 - val_loss: 0.9170 - val_accuracy: 0.7200
Epoch 96/100
10/10 [=====] - 3s 294ms/step - loss: 0.6732 - accuracy: 0.7056 - val_loss: 0.6386 - val_accuracy: 0.6200
Epoch 97/100
10/10 [=====] - 3s 306ms/step - loss: 0.7696 - accuracy: 0.7088 - val_loss: 0.7705 - val_accuracy: 0.6200
Epoch 98/100
10/10 [=====] - 3s 295ms/step - loss: 0.6203 - accuracy: 0.7277 - val_loss: 0.6863 - val_accuracy: 0.6800
Epoch 99/100
10/10 [=====] - 3s 296ms/step - loss: 0.5923 - accuracy: 0.6902 - val_loss: 0.7211 - val_accuracy: 0.7200
Epoch 100/100
10/10 [=====] - 3s 282ms/step - loss: 0.6554 - accuracy: 0.7439 - val_loss: 0.9390 - val_accuracy: 0.6800

In [9]: print("Model Runtime (seconds):",end - start)

Model Runtime (seconds): 296.63947343826294
```

For task 1, running on local CPU takes around 16seconds per epoch and 2s/step totaling up to 748 seconds. However, when running on the SCC using 1 GPU, the time for each epoch decreased to 3seconds and approx. 300ms/step, decreasing total run time to 127seconds.

Similarly, for task 2, running on local CPU takes around 10seconds per epoch and 1s/step totaling up to 1060 seconds. However, when running on the SCC using 1 GPU, the time for each epoch decreased to 3seconds and approx. 300ms/step, decreasing total run time to 296seconds.

This proves that running on a GPU drastically improves the training time of neural network models.