Configuration Manual

MSc Research Project
Data Analytics

Saurabh Thakare
Student ID: x20196415

School of Computing
National College of Ireland

Supervisor: Dr. Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saurabh Thakare
Student ID:	x20196415
Programme:	Msc. Data Analytics
Year:	2021/2022
Module:	Msc. Research Project
Supervisor:	Dr. Jorge Basilio
Submission Due Date:	15/08/2022
Project Title:	Knowledge Distillation of the ResNet50 Model for Ocular Diseases Analysis
Word Count:	1232
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Saurabh Thakare
Date:	12th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Saurabh Babarao Thakare
x20196415

1 Introduction

The research is related to the knowledge distillation of the Resnet50 model for ocular diseases analysis. All of the information pertaining to the hardware and software used at the time of conducting research is contained in this configuration manual. Additionally, every library, class, function, and script used in doing the research is explained in this manual. This manual includes snapshots that demonstrate each stage, including how to execute the research code, how to set up an environment, and how to install suitable packages. Additionally, snapshots covered in this manual were data import procedures, image analysis techniques, data augmentation, and transformation procedures, and research verification procedures.

There are five sections in this configuration manual. The hardware requirements are explained in the first section, the data gathering is explained in the second, the data analysis is explained in the third, the model specification is explained in the fourth, and the model evaluation and comparison are explained in the final section.

2 Environmental Setup

The hardware and software utilized to conduct the research are described in this section.

2.1 Hardware Requirements

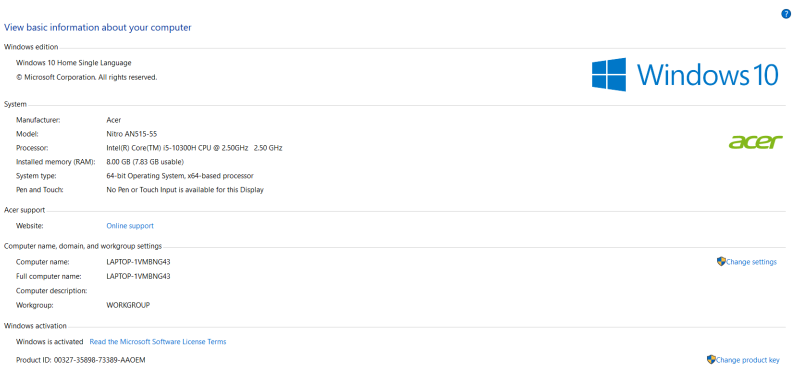

Figure 1: Hardware Configuration

As can be seen from the snapshot (figure 1) that the processor used for performing the research is the 10th Generation Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz, with 8 GB (RAM) of installed memory, 237GB SSD, and a 64-bit Windows 10 operating system.

2.2 Virtual Environmental

The 'thesiskernel' virtual environment is created for this research. The command is used to create VM. To create in Jupyter Notebook following steps have been taken:

1. I opened the command prompt by entering cmd in the windows search option and entered the 'cd' command with the directory named 'D:/NCI/Research in Computing/' to change the directory of the folder.
2. To create a new virtual environment (python -m venv Thesis) command is used.
3. To link this environment with the Jupyter notebook kernel, the following command is used: ipython kernel install –user –name = thesiskernel.
4. Finally activated the environment with the command: Thesis\Scripts\activate.

Figure 2 shows the virtual environment in the command prompt.

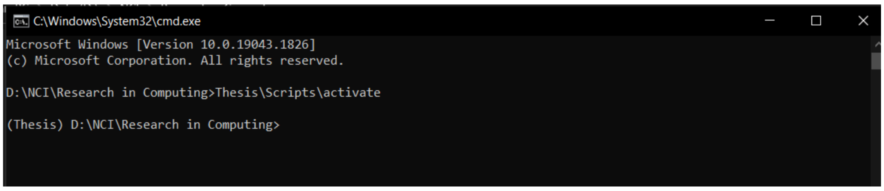

Figure 2: Virtual Environment Setup

All the installed libraries are stored in the Requirements.txt file using the commands:

$$pipfreeze > Requirements.txt$$

Figure 3: Requirement File Creation Command

This requirement file gives all libraries and packages with their version related to this research environment setup. All libraries and packages can be installed using one command:
pip install -r Requirements.txt

This research is using the Jupyter Notebook to run code and visualize the output of the research.From figure 4 it can be seen that the 'thesiskernel' environment has been used to perform this research.

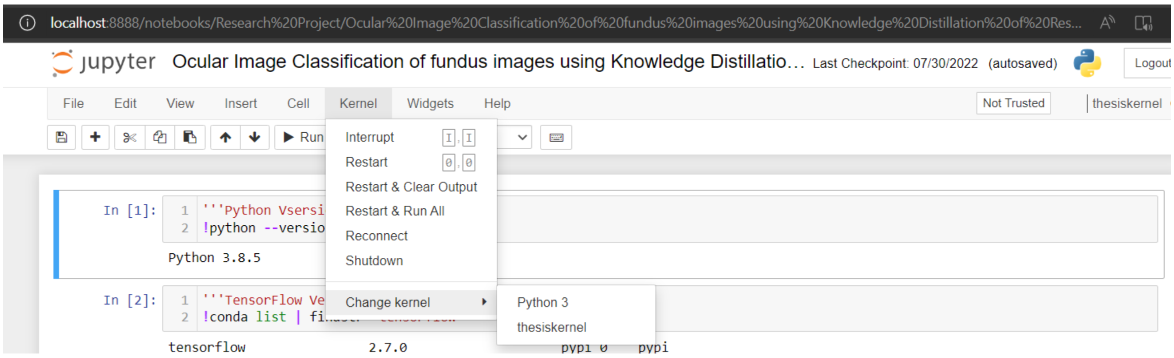

Figure 4: Jupyter Nootebook Kernel

2.3 Software Specifications

Software with Versions:

Anaconda(anaconda3) Version conda 4.13.0
Jupyter Notebook Version 6.1.4
Python Version 3.8.5
TensorFlow Version 2.7.0

3 Data Gathering

The data relating to multiple labeled ocular diseases fundus images are collected from the Kaggle website:

https://www.kaggle.com/datasets/jr2ngb/cataractdataset

4 Data Analysis

This section provides a snapshot of each piece of code used to accomplish the data analysis task on images of ocular diseases.

4.1 Importing Libraries

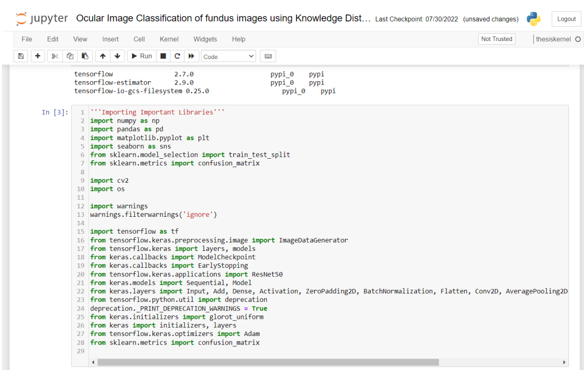

Figure 5: Importing Libraries

Different libraries, including NumPy, Pandas, Sklearn, Tensorflow, Keras, OpenCV, Matplotlib, Seaborn, etc., were used to conduct the research. Figure 5 lists each of these libraries.

4.2 Analyzing Images and Classes

Using the snapshot code mentioned above, the initialization of the dataset path and the total number of images have been determined. Figure 6 shows the dataset path initialization and visualizes the number of images in each class.

```
In [170]:
'''Class Names'''
class_name=dataset.class_names
class_name

Out[170]: ['1_normal', '2_cataract', '2_glaucoma', '3_retina_disease']

In [171]:
'''Describimg the Image Batch and Label Batch'''
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy)

(32, 256, 256, 3)
<bound method _EagerTensorBase.numpy of <tf.Tensor: shape=(32,), dtype=int32, numpy=
array([3, 2, 0, 2, 0, 3, 2, 1, 0, 3, 2, 3, 0, 2, 0, 2, 0, 1, 1, 1, 1, 3,
       3, 2, 0, 3, 2, 2, 1, 0, 0, 1])>>
```

Figure 6: Describing the Labels and Tensors

4.3 Applying Data Augmentation Method

```
In [28]:
balanced_dataset_folder=r'balanced_dataset'

In [62]:
'''Creating the balance dataset using flipping and rotating'''
c=0
List = os.listdir(DATA_PATH)
for x in List:
    folder= balanced_dataset_folder+r'/'+x+r'/'
    if os.path.exists(folder):
        pass
    else:
        os.makedirs(folder)
    path=os.path.join(DATA_PATH,x)
    images_folder = os.listdir(path)
    for images in images_folder:
        image_path = path + '/' + images
        if x == '1_normal':
            image= cv2.imread(image_path)
            cv2.imwrite(folder+x+str(c)+r'_.png', image)
            c+=1
        else:
            image1 = cv2.imread(image_path)
            cv2.imwrite(folder+x+str(c)+r'_.png',image1)
            c+=1

            image2 = cv2.flip(image1,1)
            cv2.imwrite(folder+x+str(c)+r'_.png',image2)
            c+=1

            image3 = cv2.rotate(image1,cv2.ROTATE_180)
            cv2.imwrite(folder+x+str(c)+r'_.png',image3)
            c+=1
print('Count of total images in balanced dataset is: {}'.format(c))

Count of total images in balanced dataset is: 1203
```

Figure 7: Data Augmentation

Since there are 300 images of normal, 101 images of glaucoma, and 100 images of each retinal and cataract in the original dataset. The initial dataset needs balance. Two augmentations methods: flipping and rotating are utilized to balance it. The augmentation code used on the unbalanced dataset is displayed in Figure 7. The cv2.flip() function's 1 parameter specifies vertical flipping and the cv2.ROTATE_180 parameter specifies the 180° clockwise rotation of the images. The balanced_dataset_folder path is where the balanced data is kept.

4.4 Visualizing Balanced Dataset

```
In [31]:
folders = os.listdir(balanced_dataset_folder)
train_number = []
class_num = []

for folder in folders:
    train_files = os.listdir(balanced_dataset_folder + '/' + folder)
    train_number.append(len(train_files))
    class_num.append(folder)

zipped_lists = zip(class_num,train_number)
sorted_pairs = sorted(zipped_lists,reverse=True)
tuples = zip(*sorted_pairs)
balanced_Dataset=pd.DataFrame(sorted_pairs,columns=['Ocular Diseases','Count'])

plt.figure(figsize=(10,5))
ax=sns.barplot(x='Ocular Diseases', y='Count',data=balanced_Dataset,ci=0, palette="Blues_d")
for p in ax.patches:
    height = p.get_height()
    ax.text(x = p.get_x()+(p.get_width()/2),y = height+0.9, s = '{:.0f}'.format(height),ha = 'center')
ax.set_title("Balanced Dataset")
plt.show()
```

Figure 8: Balanced Dataset Visulization

The number of images in a balanced dataset is shown by the code in figure 8. A balanced dataset had 303 images of glaucoma and 300 images of each normal, cataract, and retinal disease.

4.5 Images Transformation and Storing Images in Tensors

```
In [65]:
'''Initilizing parameters related to images'''
IMAGE_WIDTH= 256
IMAGE_HEIGHT= 256
BATCH_SIZE= 32
CHANNEL= 3
SEED=123
num_classes=4
balanced_dataset_folder=r'balanced_dataset'

In [169]:
'''Defining Dataset'''
dataset=tf.keras.preprocessing.image_dataset_from_directory(balanced_dataset_folder,
                                                            image_size= (IMAGE_WIDTH,IMAGE_HEIGHT),
                                                            shuffle= True,
                                                            batch_size=BATCH_SIZE,
                                                            crop_to_aspect_ratio=True,
                                                            seed=SEED
                                                            )
Found 1203 files belonging to 4 classes.
```

Figure 9: Dataset Importing and Resizing

once the balanced folder has been created, balanced images are created and saved there. Using the 'image_dataset_from_directory' Keras library, both balanced and unbalanced images have been imported into a Jupyter notebook. Figure 9 shows that the image's width and height are taken to be 256, the batch size is taken to be 32, and the images have been cropped and shuffled. At the time of importing the images, the images were resized and cropped.

4.6 Describing the Images classes and Batches

```
In [170]:
'''Class Names'''
class_name=dataset.class_names
class_name
Out[170]: ['1_normal', '2_cataract', '2_glaucoma', '3_retina_disease']

In [171]:
'''Describimg the Image Batch and Label Batch'''
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy)
(32, 256, 256, 3)
<bound method _EagerTensorBase.numpy of <tf.Tensor: shape=(32,), dtype=int32, numpy=
array([3, 2, 0, 2, 0, 3, 2, 1, 0, 3, 2, 3, 0, 2, 0, 2, 0, 1, 1, 1, 1, 3,
       3, 2, 0, 3, 2, 2, 1, 0, 0, 1])>>
```

Figure 10: Describing the Dataset

The total number of classes and the manner in which the images are distributed in batches are described by the code in Figure 10. Additionally, it explains the tensor structure, size, and data type.

4.7 Visualizing the Transformed Images

```
In [172]:
plt.figure(figsize=(8, 9))
for image_batch, label_batch in dataset.take(1):
    for x in range(9):
        ax=plt.subplot(3,3,x+1)
        plt.imshow(image_batch[x].numpy().astype('int'))
        plt.title(class_name[label_batch[x]])
#         plt.axis('off')
```

Figure 11: Transformed Images Visualization

The 9 resized and cropped images are visualized by the code in figure 11. The 3x3 matrix is used to display the images. The images are selected randomly.

4.8 Splitting of Dataset

The code in figure 12 is splitting the dataset into training and testing samples. The ratio used is 8:2. Which means 80% of the dataset was used for training purposes and 20% of the dataset is used for testing purposes.

```
In [173]:
'''Dividing dataset into train and validation (80%, 20%)'''

print("Length of 1-batch of dataset: {}".format(len(dataset)))

'''Training dataset'''
train_data_batch=int(len(dataset)*0.80)
train_data=dataset.take(train_data_batch)
print("Length of 1-batch of train data: {}".format(len(train_data)))

'''Testing dataset'''
test_data=dataset.skip(train_data_batch)
print("Length of 1-batch of test data: {}".format(len(test_data)))

Length of 1-batch of dataset: 38
Length of 1-batch of train data: 30
Length of 1-batch of test data: 8
```

Figure 12: Splitting Dataset

5 Model Architecture Implementation and Compilation

The code needed to implement the model is explained in this section.

5.1 ResNet50 Model Architecture (Teacher Model)

```
In [119]:
'''Creating fuction of teacher model architecture '''
def Teacher_Model(input_shape=(IMAGE_WIDTH,IMAGE_HEIGHT,3)):
    teacher = ResNet50(weights="imagenet", include_top = False, input_shape=input_shape)
    model = Sequential()
    model.add(teacher)
    model.add(Flatten())
    model.add(Dense(units=512,activation='relu'))
    model.add(Dense(num_classes,activation="softmax"))

    return model

In [120]:
'''Summary of teacher model'''

teacher_model=Teacher_Model()
teacher_model.summary()
```

Figure 13: ResNet50 Model Architecture

The ResNet50 network is described in the code in Figure 13. The 'ImageNet' pre-trained weight is used by the ResNet50 model. The research uses the ResNet50 as a teacher model. The model is trained using SoftMax activation and the 4 classes of ocular diseases. The Summary of the model using teacher_model.summary() function.

5.2 ResNet50 Compilation

```
In [121]:
'''Compilation of teacher model'''
teacher_model.compile(optimizer=tf.keras.optimizers.Adam(),
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])
```

Figure 14: ResNet50 Model Compilation

The compilation code of the ResNet50 model and training part is explained in figure 14 and figure 15. The model is trained with 32 batch sizes and 25 epoch sizes.

```
In [125]:
'''Model history'''
history= teacher_model.fit(train_data,batch_size=BATCH_SIZE,epochs= 25, validation_data=test_data)
```

Figure 15: ResNet50 Model Training

5.3 CNN12 Model Architecture (Student Model)

Conv2D, MaxPool, and Dropout are applied to generate the CNN12 model. Conv2D's 6 layers were each 2 layers processed through a different filter. Figure 16 displays CNN12's model architecture code as well as a summary of it.

```
In [175]:
def Student_model(input_shape=(IMAGE_WIDTH,IMAGE_HEIGHT,3)):
    model=tf.keras.models.Sequential()

    model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='relu',input_shape=input_shape))
    model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),padding='valid',activation='relu'))
    model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
    model.add(tf.keras.layers.Dropout(rate=0.2))

    model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='relu'))
    model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=(3,3),padding='valid',activation='relu'))
    model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
    model.add(tf.keras.layers.Dropout(rate=0.2))

    model.add(tf.keras.layers.Conv2D(filters=128,kernel_size=(3,3),padding='same',activation='relu'))
    model.add(tf.keras.layers.Conv2D(filters=128,kernel_size=(3,3),padding='valid',activation='relu'))
    model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
    model.add(tf.keras.layers.Dropout(rate=0.2))

    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(units=512,activation='relu'))
    model.add(tf.keras.layers.Dropout(rate=0.5))
    model.add(tf.keras.layers.Dense(num_classes,activation='softmax'))

    return model

In [176]:
'''Summary of student model'''
student_model=Student_model()
student_model.summary()
```

Figure 16: CNN12 Model Architecture

5.4 CNN12 Compilation

```
In [139]:
'''Compiling student model from scratch'''
student_scratch.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)
```

Figure 17: CNN12 Compilation

Figures 17 explain the CNN12 model's compilation code and training part. 20 epoch sizes and 32 batch sizes are used to train the model.

5.5 Distiller Model Architecture

Figure 18 code shows how this research made the distiller class. This research used the distiller class to balance the ResNet50 model's logit probability with the CNN12 model logit probability. The temperature parameter, which is an optimizing parameter utilized in this class, aids the CNN12 model in achieving accuracy comparable to the ResNet50 model. Distillation loss is the difference in logit between the CNN12 model and the ResNet50 model. We need to change the temperature parameters in order to reduce distillation loss.

5.6 Distiller Model Compilation

Figures 19 explain the CNN12 distiller model's compilation code and training part. 20 epoch sizes and 32 batch sizes are used to train the model.

```python
In [174]:
class Distiller(tf.keras.Model):
    def __init__(self, student, teacher):
        super(Distiller, self).__init__()
        self.teacher = teacher
        self.student = student

    def compile(
        self,
        optimizer,
        metrics,
        student_loss_fn,
        distillation_loss_fn,
        alpha=0.1,
        temperature=3,
    ):
        """ Configure the distiller.

        Args:
            optimizer: Keras optimizer for the student weights
            metrics: Keras metrics for evaluation
            student_loss_fn: Loss function of difference between student
                predictions and ground-truth
            distillation_loss_fn: Loss function of difference between soft
                student predictions and soft teacher predictions
            alpha: weight to student_loss_fn and 1-alpha to distillation_loss_fn
            temperature: Temperature for softening probability distributions.
                Larger temperature gives softer distributions.
        """
        super(Distiller, self).compile(optimizer=optimizer, metrics=metrics)
        self.student_loss_fn = student_loss_fn
        self.distillation_loss_fn = distillation_loss_fn
        self.alpha = alpha
        self.temperature = temperature

    def train_step(self, data):
        # Unpack data
        x, y = data

        # Forward pass of teacher
        teacher_predictions = self.teacher(x, training=False)

        with tf.GradientTape() as tape:
            # Forward pass of student
            student_predictions = self.student(x, training=True)

            #Compute losses
            student_loss = self.student_loss_fn(y, student_predictions)

            # The magnitudes of the gradients produced by the soft targets scale
            # as 1/T^2, multiply them by T^2 when using both hard and soft targets.
            distillation_loss = (
                self.distillation_loss_fn(
                    tf.nn.softmax(teacher_predictions / self.temperature, axis=1),
                    tf.nn.softmax(student_predictions / self.temperature, axis=1),
                )
                * self.temperature**2
            )

            loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss

        # Compute gradients
        trainable_vars = self.student.trainable_variables
        gradients = tape.gradient(loss, trainable_vars)

        # Update weights
        self.optimizer.apply_gradients(zip(gradients, trainable_vars))

        # Update the metrics configured in `compile()`.S
        self.compiled_metrics.update_state(y, student_predictions)

        # Return a dict of performance
        results = {m.name: m.result() for m in self.metrics}
        results.update(
            {"student_loss": student_loss, "distillation_loss": distillation_loss}
        )
        return results

    def test_step(self, data):
        # Unpack the data
        x, y = data

        # Compute predictions
        y_prediction = self.student(x, training=False)

        # Calculate the loss
        student_loss = self.student_loss_fn(y, y_prediction)

        # Update the metrics.
        self.compiled_metrics.update_state(y, y_prediction)

        # Return a dict of performance
        results = {m.name: m.result() for m in self.metrics}
        results.update({"student_loss": student_loss})
        return results
```

Figure 18: Distiller Model Architecture

```python
In [178]:
'''Intilizing Distiller Class'''
distiller = Distiller(student=student_model, teacher=teacher_model)

In [192]:
'''Compiling student model using distiller class'''
distiller.compile(
    optimizer=tf.keras.optimizers.Adam(),
    metrics=['accuracy'],
    student_loss_fn=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    distillation_loss_fn=tf.keras.losses.KLDivergence(),
    alpha=0.1,
    temperature=100,
)

In [193]:
'''Trasfering teacher information to student'''
distiller_hist=distiller.fit(train_data,batch_size=BATCH_SIZE,epochs= 20, validation_data=test_data)
```

Figure 19: Distiller Model Compilation

6 Model Evaluation and Comparison

The entire code in this section is used to compare and visualize balanced versus unbalanced dataset models.

6.1 Models Evaluation and Visualization

```python
In [196]:
plt.figure(figsize=(10,10))

'''Summarizing distiller history of accuracy'''
plt.subplot(211)
plt.plot(distiller_hist.history['accuracy'])
plt.plot(distiller_hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='lower right')

'''Summarizing distiller history of loss'''
plt.subplot(212)
plt.plot(distiller_hist.history['student_loss'])
plt.plot(distiller_hist.history['val_student_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper right')
plt.show()
plt.tight_layout()
```

Figure 20: Accuracy and Loss Visualization

6.2 Ocular Diseases Datasets Result Analysis

In [197]:
```
'''Evaluating distiller test data'''
acc,loss = distiller.evaluate(test_data)
print('test loss = ', loss)
print('test accuracy = ',acc)
```

Figure 21: Test Accuracy Evaluation

In [4]:
```
'''Unbalanced dataset Model'''

dict_Unbalanced = {
                  'Models':['Teacher','Distiller(t=50)','Distiller(t=70)','Distiller(t=90)','Distiller(t=100)','Student'],
                  'Training Accuracy':[0.689,0.497,0.561,0.683,0.979,0.504],
                  'Test Accuracy':[0.691,0.471,0.568,0.719,0.966,0.495]
                 }
```

In [5]:
```
'''Structuring of different parameters with accuracy'''
Unbalanced_dataframe = pd.DataFrame(dict_Unbalanced)
Unbalanced_dataframe
```

Out[5]:

	Models	Training Accuracy	Test Accuracy
0	Teacher	0.689	0.691
1	Distiller(t=50)	0.497	0.471
2	Distiller(t=70)	0.561	0.568
3	Distiller(t=90)	0.683	0.719
4	Distiller(t=100)	0.979	0.966
5	Student	0.504	0.495

Figure 22: Unbalanced Dataset Results

In [7]:
```
'''Balanced dataset Model'''

dict_balanced = {
                  'Models':['Teacher','Distiller(t=50)','Distiller(t=70)','Distiller(t=90)','Distiller(t=100)','Student'],
                  'Training Accuracy':[0.821,0.535,0.610,0.710,0.823,0.567],
                  'Test Accuracy':[0.777,0.514,0.609,0.637,0.765,0.526]
                 }
```

In [8]:
```
'''Structuring of different parameters with accuracy'''
Balanced_dataframe = pd.DataFrame(dict_balanced)
Balanced_dataframe
```

Out[8]:

	Models	Training Accuracy	Test Accuracy
0	Teacher	0.821	0.777
1	Distiller(t=50)	0.535	0.514
2	Distiller(t=70)	0.610	0.609
3	Distiller(t=90)	0.710	0.637
4	Distiller(t=100)	0.823	0.765
5	Student	0.567	0.526

Figure 23: Balanced Dataset Results

In [11]:
```
'''Comparison between testing accuracy of balanced and unbalanced datasets'''
plt.figure(figsize=(12,5))
ax=sns.lineplot(x='Models',y='Test Accuracy',data=Unbalanced_dataframe,marker='o')
ax=sns.lineplot(x='Models',y='Test Accuracy',data=Balanced_dataframe, marker='o')
ax.set_title('Balanced V/s Unbalanced Dataset')
ax.set_ylabel('Test Accuracy')
plt.legend(['Unbalnced', 'Balanced'], loc='upper right')
ax.set_xticklabels(labels=['Teacher','Distiller(t=50)','Distiller(t=70)','Distiller(t=90)','Distiller(t=100)','Student'],
                   rotation=45)
plt.show()
```

Figure 24: Comparison Between Balanced and Unbalanced Results

Repositories

1.1 GitHub Repository

https://github.com/SaurabhNCI/Reseach-Project/tree/main

1.2 Kaggle Dataset Repository

https://www.kaggle.com/datasets/jr2ngb/cataractdataset

References

Below links were used to resolve the issues related to the research.

1. https://keras.io/examples/vision/knowledge_distillation/
2. https://www.tensorflow.org/api_docs/python/tf/keras/