Pattern Final Project

Submitted by:

1. **Rokaia Mohamed**
2. **Hendwan Abozide**
3. **Youssef Saleh**

Problem Statement

Our problem is to classify the input images into their correct classes by implementing a **convolutional neural network (CNN)** from scratch by implementing **CNN** using **Keras**.

By using **Keras**, we need to add CNN layers and determine how each layer affects our model and through practice, find the best built-model for our two datasets and achieve the highest accuracy we can.

In order to test how efficient our model is and if it is able to classify the images correctly, we compare our model with a **transfer learning model**. We used the two datasets; one containing black and white cell images and the second one containing general random colored images.

Method & Algorithms

Keras Model General Specifications

- We chose the following **10** classes for the implementation:
  - **motorbikes**
  - **helicopter**
  - **airplanes**
  - **brain**
  - **car_side**
  - **panda**
  - **chandelier**
  - **cougar_face**
  - **crab**
  - **grand_piano**

- We standardized our input images as it is preferred for the neural network to have the input images in this format. (CNN models prefer input values between -1 to 1)

- We used the same **x_test** and **y_test** for both models.

- We made **two** models; one using **Data Augmentation** and the other without. Data augmentation in our model reads the images in each epoch as if they are a new dataset, as it, in our case; it shifts the width and height with a specific value and also rotates the image by a specific degree. In a nutshell, it is like generating new images in each epoch while they are actually the same ones and this can contribute in the accuracy results of our model.

- **Optimizer used:** We used **Adam** optimizer as a gradient descent method for changing the weights and minimizing the loss. We tried **RMSprop** also, but Adam worked better in our model.

- We used **EarlyStopping** function during training. It is used to monitor a specific variable. We made it to check on **val_loss.** The function takes a parameter called **patience**; which is a number that specifies the number of epochs needed to check on **val_loss**, if no change occurred to **val_loss** for a repetitive number of epochs, the training stops.

- We used **ModelCheckpoint** function during training. It is used to monitor a specific variable also. We used it to monitor the **val_accuracy.** In each epoch. It checks if the value of **val_accuracy** changed from the previous epoch, if yes it saves the model at the current epoch and continue the training.

- ***Keras Model Layers***

1. Cell Images Model

- We used **4 convolutional layers**, with neurons number starting with **32** and increasing till **256** in the last layer.
- After each convolutional layer, a **MaxPool** is added with varying sizes. The first 3 layer has size 2 X 2 and the last layer is of size 3 X 3.
- No **Dropout** layers were added in this model, as data is already small. When added, the model's accuracy became worse.

2. General Random Images Model

- We used **5 convolutional layers**, with neurons numbers starting with **16** and increasing till **256** in the last layer.
- After each convolutional layer, a **MaxPool** layer is added, but with varying sizes. The first two max pool layers have size 2 X 2 and the last 3 were size of 3 X 3.
- We used **kernel_initializer** to randomly initialize the weights value and **kernel_regularizer** to adjust the weight values (minimize them) to avoid over fitting.
- We added only one **Dropout** layer after all convolutional and maxpool layers with value of **0.2.**
- The last **Dense** layer has value of **512**. This value worked well for our model, increasing or decreasing it may cause increase in the processing time and/or worse accuracy

Keras Model Output

In all of the outputs, we always take the best "epoch" iteration for our model using checkpoints

1. Cell Images Model

- ***Keras with Data Augmentation***

Timing and Specs of machine

Time	Specs
Maximum Epochs: 60 **Epochs reached:** 38 Each Epoch took around 10 seconds **Total Time to train**: 6.333 minutes **Size of dataset**: 862 images **Size of each image:** 382,382,1 (Greyscale)	Using MSI GL65 9SDK Laptop **GPU used:** GTX 1660Ti **GPU Memory** 6GB **Available RAM** 16 GB

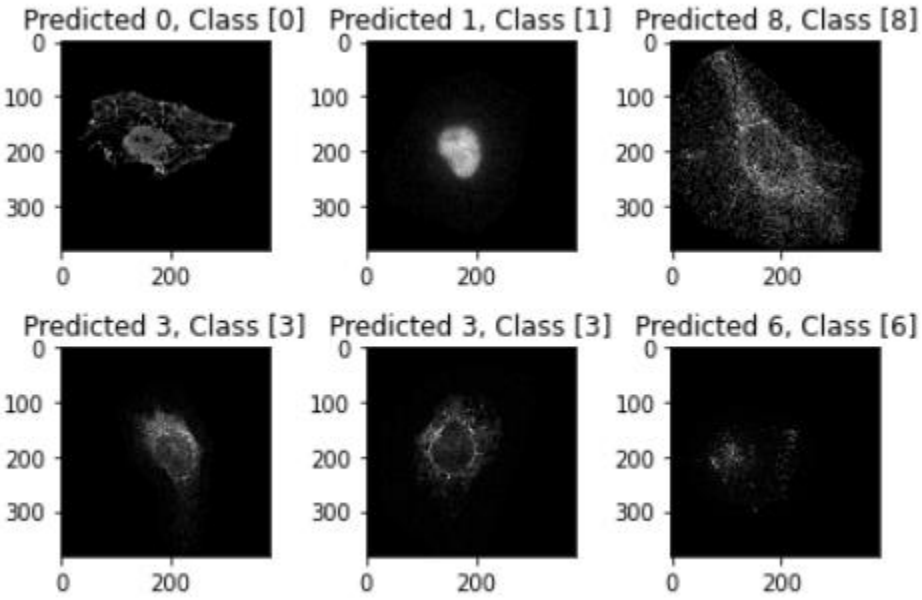
1. Evaluation output: **93%** accuracy and loss equal **0.25**

```
[==============================] - 0s 45ms/step - loss: 0.2544 - accuracy: 0.9308
```

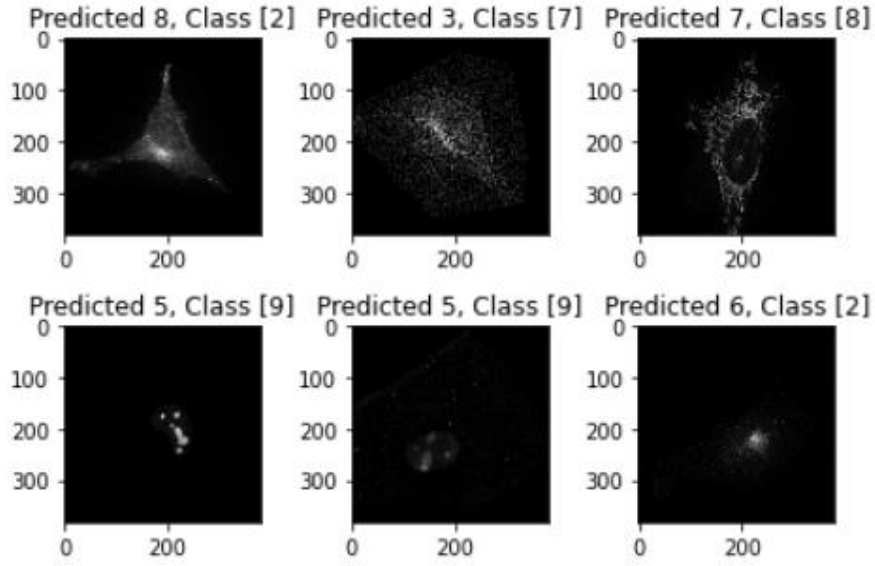
2. **Classification Report** shows that the model accuracy is **93%** and shows the recall, precision and f1-score and the number of x_test images in each class **(support)**.

	precision	recall	f1-score	support
Class: actin	1.00	1.00	1.00	20
Class: dna	1.00	1.00	1.00	16
Class: endosome	1.00	0.55	0.71	11
Class: er	0.80	1.00	0.89	12
Class: golgia	0.93	0.93	0.93	14
Class: golgpp	0.71	0.83	0.77	6
Class: lysosome	1.00	0.91	0.95	11
Class: microtubules	1.00	0.94	0.97	16
Class: mitochondria	0.80	1.00	0.89	12
Class: nucleolus	1.00	1.00	1.00	12
accuracy			0.93	130
macro avg	0.92	0.92	0.91	130
weighted avg	0.94	0.93	0.93	130

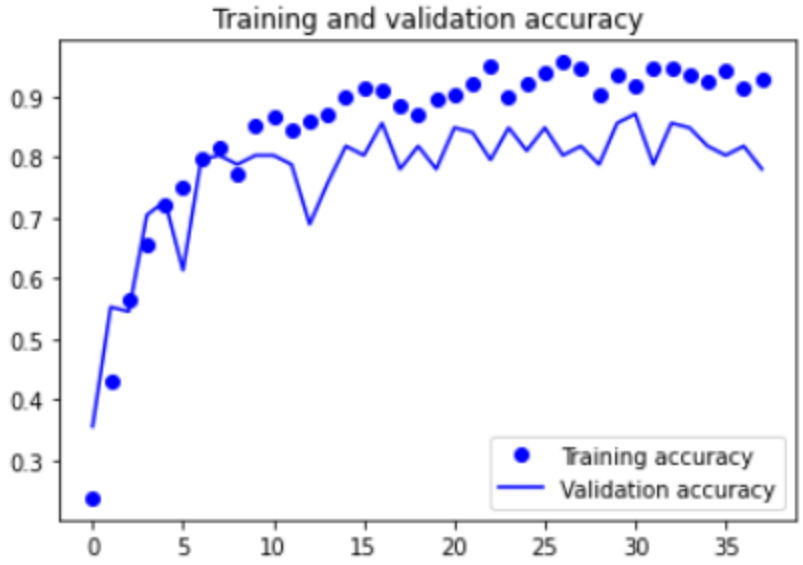
3. A Sample output of the **correctly** predicted images



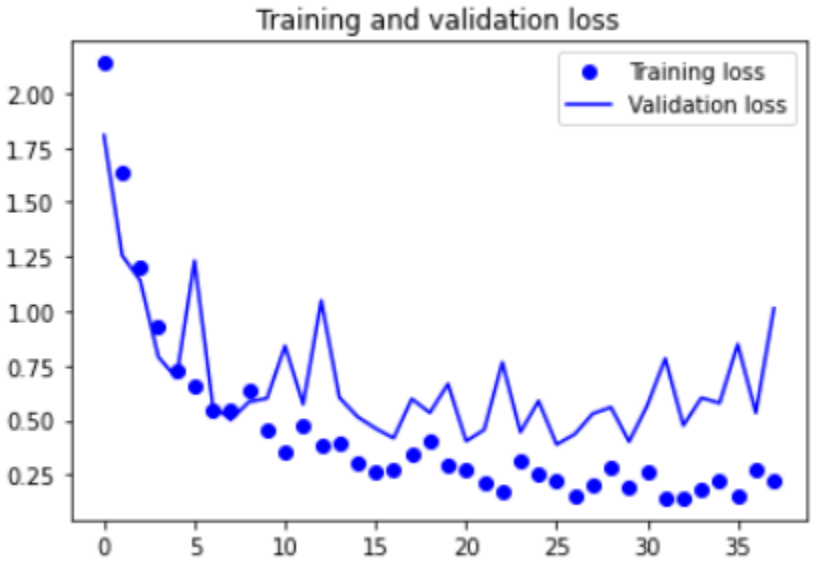
4. A Sample output of the **incorrectly** predicted images



5. Plotting **Training accuracy** with **Validation accuracy**



6. Plotting **Training loss** with **Validation loss**



- ***Keras <u>without</u> Data Augmentation***

Timing and Specs of machine

Time	Specs
Maximum Epochs: 60 **Epochs reached**: 16 Each Epoch took around 6 seconds **Total Time to train:** 1.6 minutes **Size of dataset**: 862 images **Size of each image:** 382,382,1 (Greyscale)	Using MSI GL65 9SDK Laptop **GPU used:** GTX 1660Ti **GPU Memory** 6GB **Available RAM** 16 GB

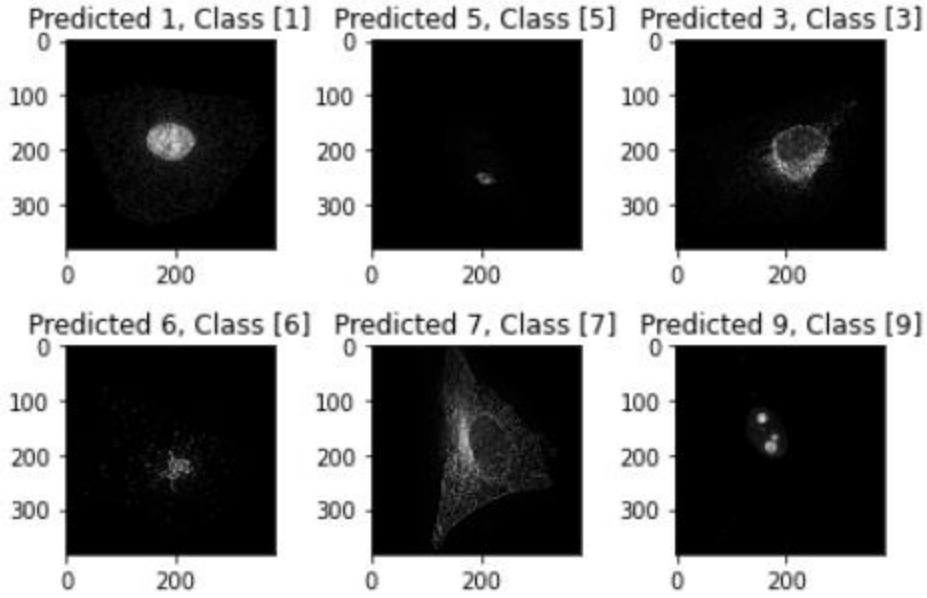
1. Evaluation output: **73%** accuracy and loss equal **1.58**

```
[==============================] - 0s 44ms/step - loss: 1.5820 - accuracy: 0.7308
```

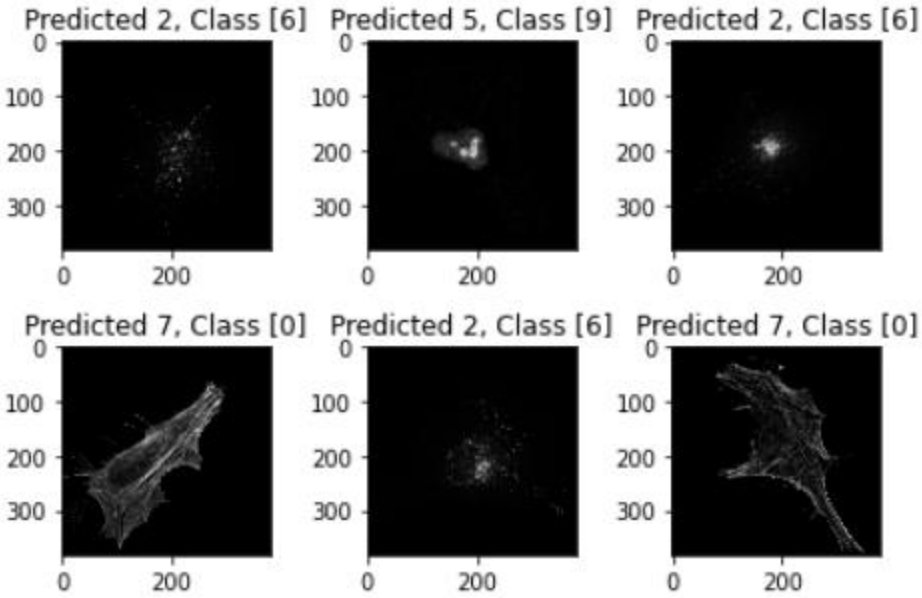
2. **Classification Report** shows that the model accuracy is **73%** and shows the recall, precision and f1-score and the number of x_test images in each class **(support)**.

	precision	recall	f1-score	support
Class: actin	0.90	0.90	0.90	20
Class: dna	0.84	1.00	0.91	16
Class: endosome	0.43	0.27	0.33	11
Class: er	0.62	0.67	0.64	12
Class: golgia	0.87	0.93	0.90	14
Class: golgpp	0.50	0.50	0.50	6
Class: lysosome	0.75	0.82	0.78	11
Class: microtubules	0.56	0.62	0.59	16
Class: mitochondria	0.75	0.50	0.60	12
Class: nucleolus	0.75	0.75	0.75	12
accuracy			0.73	130
macro avg	0.70	0.70	0.69	130
weighted avg	0.72	0.73	0.72	130

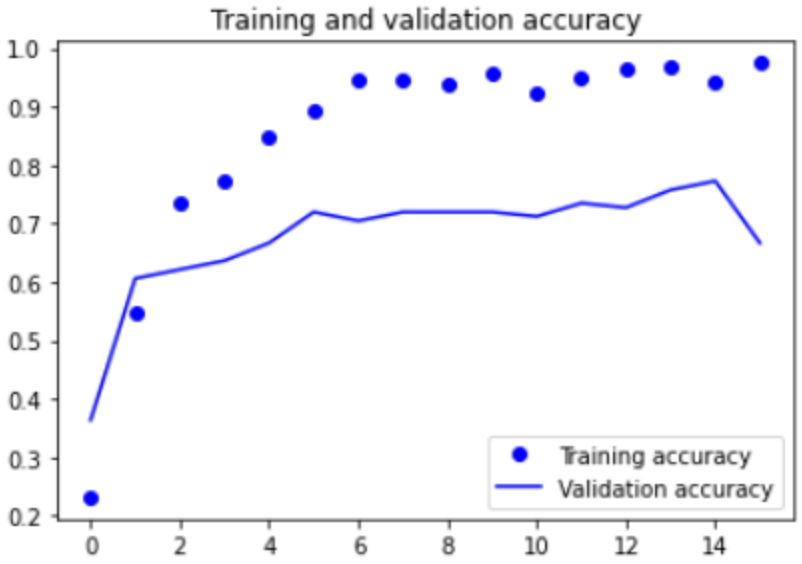
3. A Sample output of the **correctly** predicted images



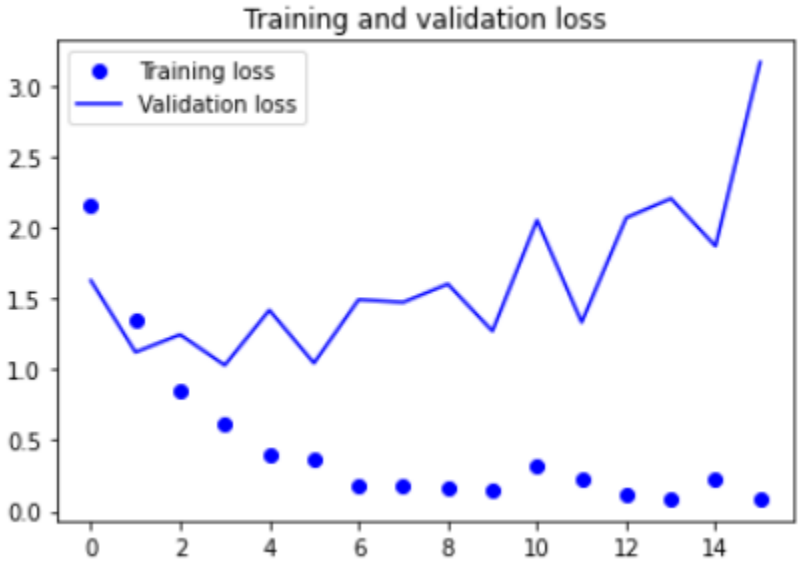
4. A Sample output of the **incorrectly** predicted images



5. Plotting **Training accuracy** with **Validation accuracy**



6. Plotting **Training loss** with **Validation loss**



- ***Output Observation for the two cells model***

  Data augmentation using the cells dataset causes a significant effect on the accuracy. Adding it causes the accuracy to increase to **93%** from **73%.**

  This is due to the fact that Keras' data augmentation allows each epoch to be a **"uniquely generated"** training set for the model (by applying the transformation functions from the datagen such as **shifting**, **zooming** and **rotating**).

  So, it solved the problem of having few data in the training and allowed the model to train **more effectively**.

2. General Random Images Model

- ***Keras <u>with</u> Data Augmentation***

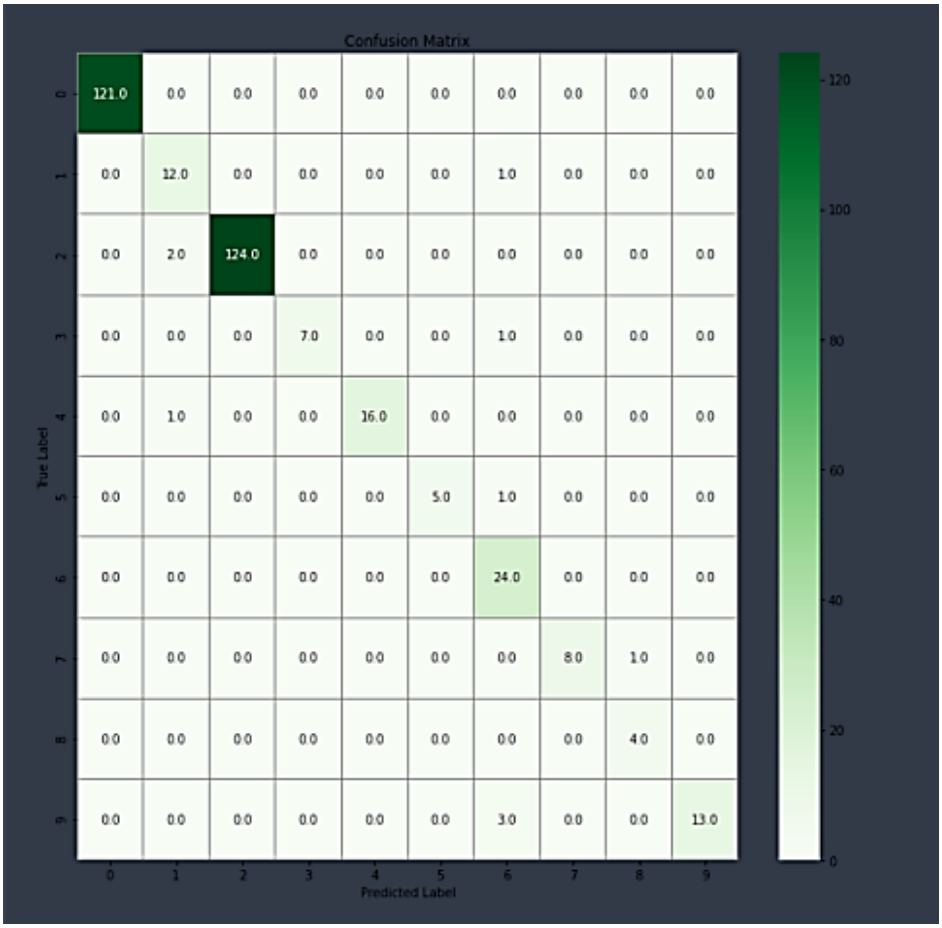
  Timing and Specs of machine

Time	Specs
Maximum Epochs: 60 **Epochs reached:** 43 Each Epoch took around 20 seconds **Total Time to train:** 14 minutes **Size of dataset:** 2293 images **Size of each image:** 256,256,3 (RGB)	Google Collab GPU runtime **GPU used:** Nvidia K80 **GPU Memory:** 12GB / 16GB **Available RAM:** 12GB (upgradable to 26.75GB)

1. Evaluation output: **97%** accuracy and loss equal **0.16**

```
11/11 [==============================] - 1s 58ms/step - loss: 0.1632 - accuracy: 0.9709
```

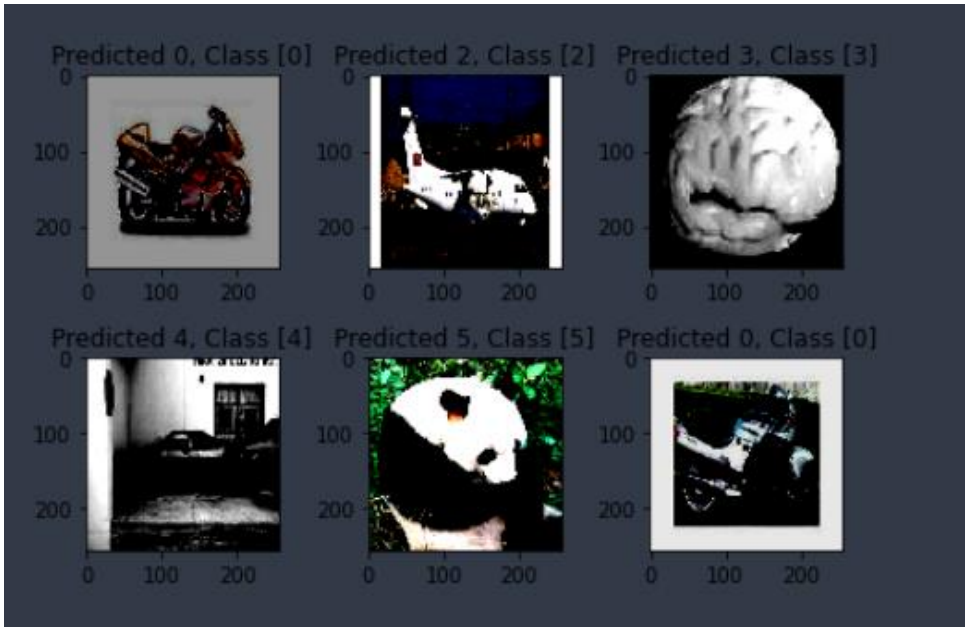
2. **Confusion matrix** shows the number of images that was predicted correctly in each class. For example, in the following output class **2**, has **124** and the model correctly predicted all of them.



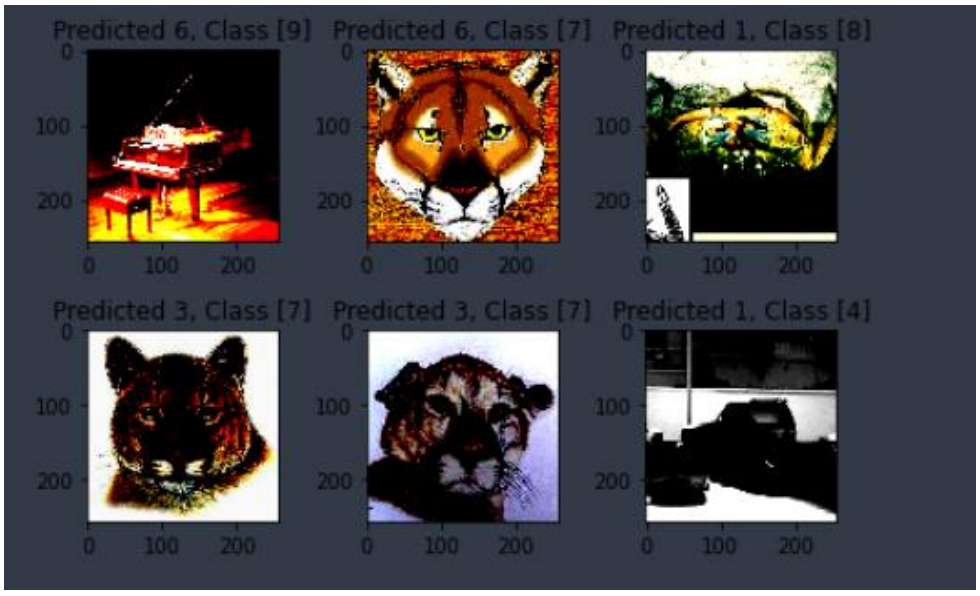
3. **Classification Report** shows that the model accuracy is **97%** and shows the recall, precision and f1-score and the number of x_test images in each class **(support)**.

	precision	recall	f1-score	support
Class: Motorbikes	1.00	1.00	1.00	121
Class: helicopter	0.80	0.92	0.86	13
Class: airplanes	1.00	0.98	0.99	126
Class: brain	1.00	0.88	0.93	8
Class: car_side	1.00	0.94	0.97	17
Class: panda	1.00	0.83	0.91	6
Class: chandelier	0.80	1.00	0.89	24
Class: cougar_face	1.00	0.89	0.94	9
Class: crab	0.80	1.00	0.89	4
Class: grand_piano	1.00	0.81	0.90	16
accuracy			0.97	344
macro avg	0.94	0.93	0.93	344
weighted avg	0.98	0.97	0.97	344

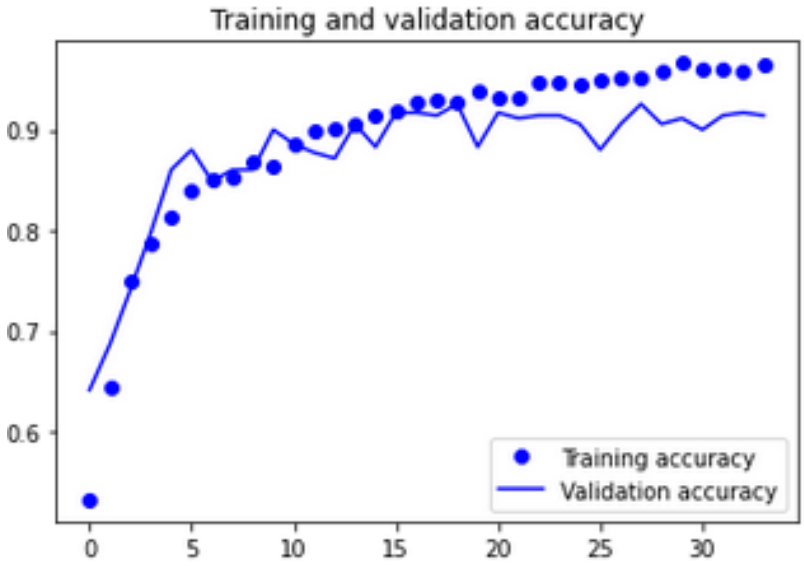
4. A Sample output of the **correctly** predicted images



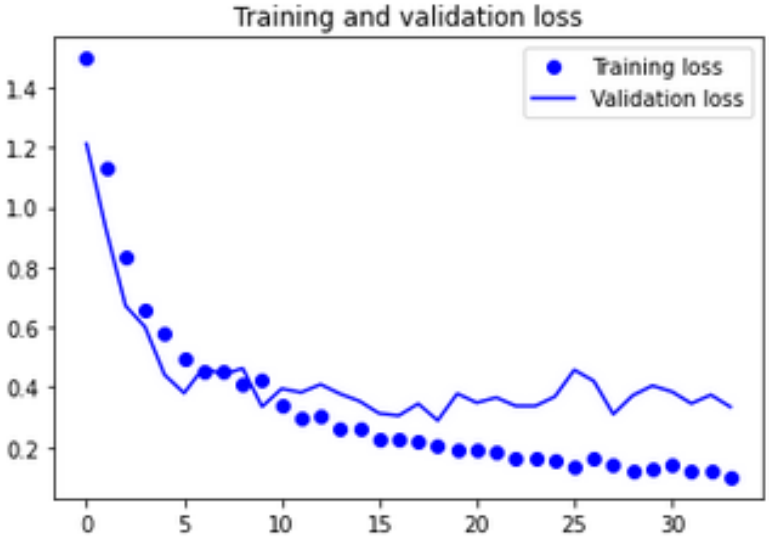
5. A Sample output of the **incorrectly** predicted images



7. Plotting **Training accuracy** with **Validation accuracy**



8. Plotting **Training loss** with **Validation loss**



- ***Keras <u>without</u> Data Augmentation***

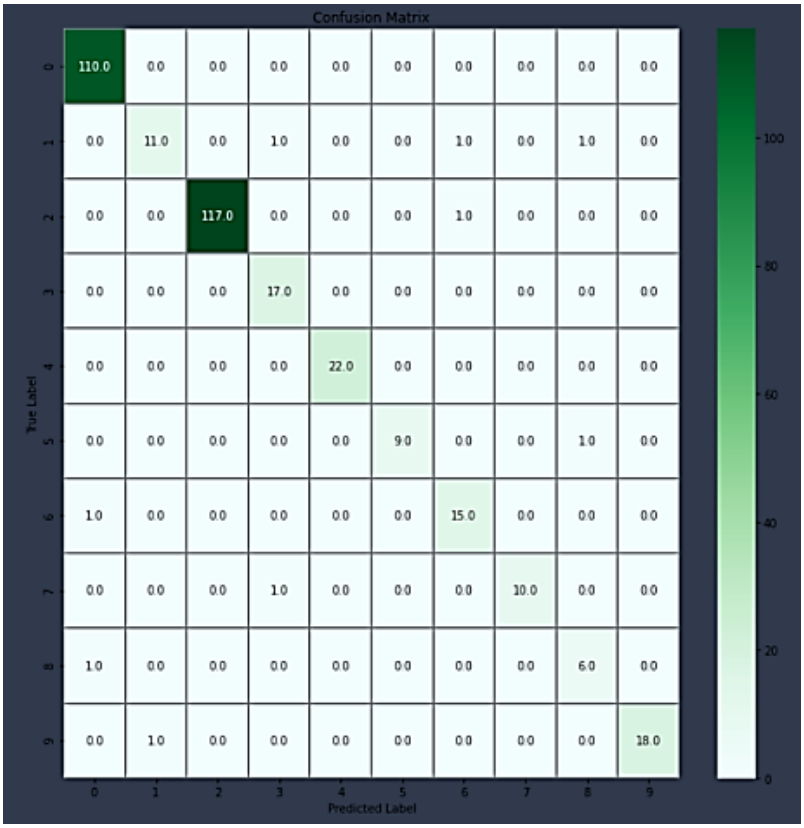
Timing and Specs of machine

Time	Specs
Maximum Epochs: 60 **Epochs reached:** 48 Each Epoch took around 3 seconds **Total Time to train:** 3 minutes **Size of dataset**: 2293 images **Size of each image**: 256,256,3 (RGB)	Google Collab GPU runtime **GPU used:** Nvidia K80 **GPU Memory:** 12GB / 16GB Available RAM 12GB (upgradable to 26.75GB)

1. Evaluation output: **97%** accuracy and loss equal **0.1**

```
[==============================] - 6s 46ms/step - loss: 0.1013 - accuracy: 0.9738
```

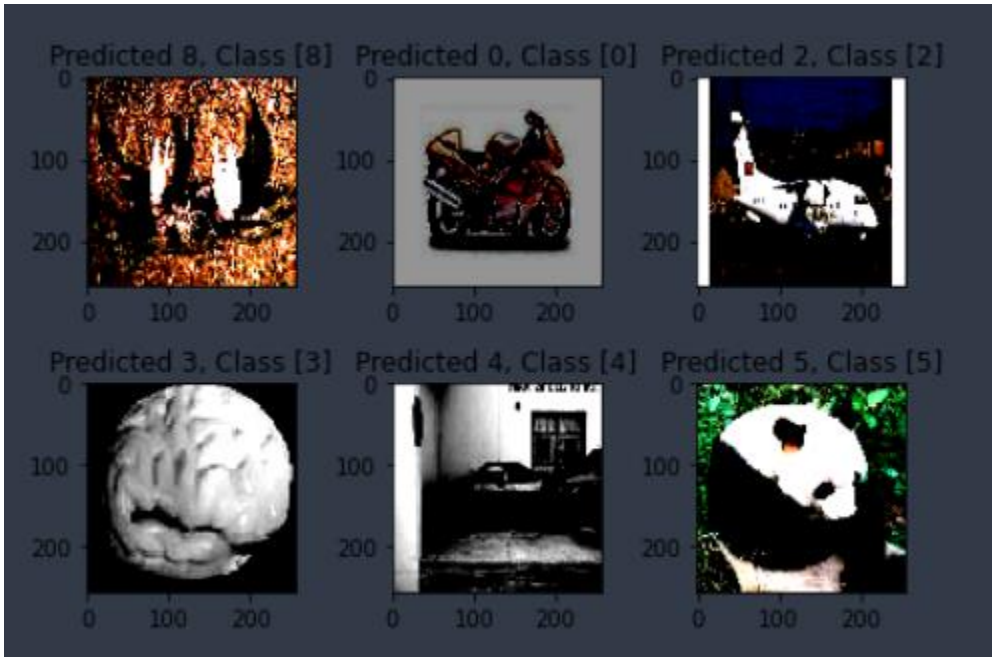
2. **Confusion matrix**



3. **Classification Report** shows that the model accuracy is **97%** and shows the recall, precision and f1-score and the number of x_test images in each class **(support)**.

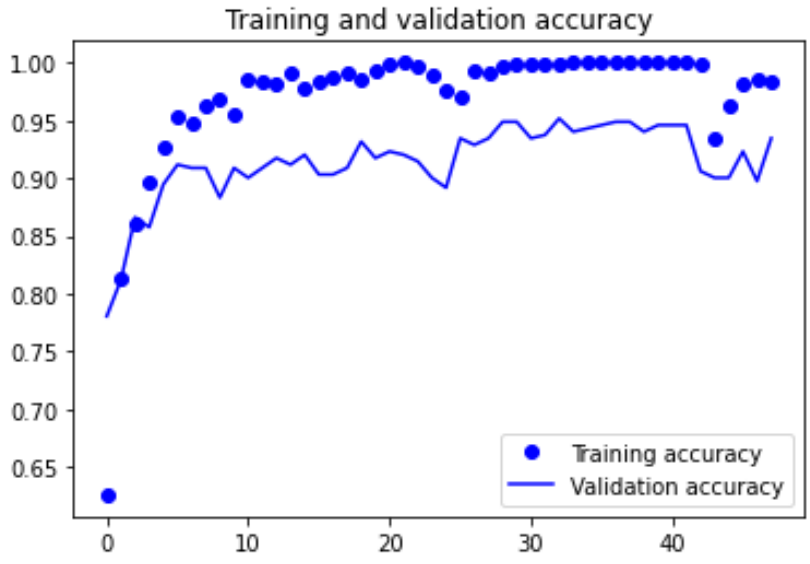
	precision	recall	f1-score	support
Class: Motorbikes	0.98	1.00	0.99	110
Class: helicopter	0.92	0.79	0.85	14
Class: airplanes	1.00	0.99	1.00	118
Class: brain	0.89	1.00	0.94	17
Class: car_side	1.00	1.00	1.00	22
Class: panda	1.00	0.90	0.95	10
Class: chandelier	0.88	0.94	0.91	16
Class: cougar_face	1.00	0.91	0.95	11
Class: crab	0.75	0.86	0.80	7
Class: grand_piano	1.00	0.95	0.97	19
accuracy			0.97	344
macro avg	0.94	0.93	0.94	344
weighted avg	0.98	0.97	0.97	344

4. A Sample output of the **correctly** predicted images

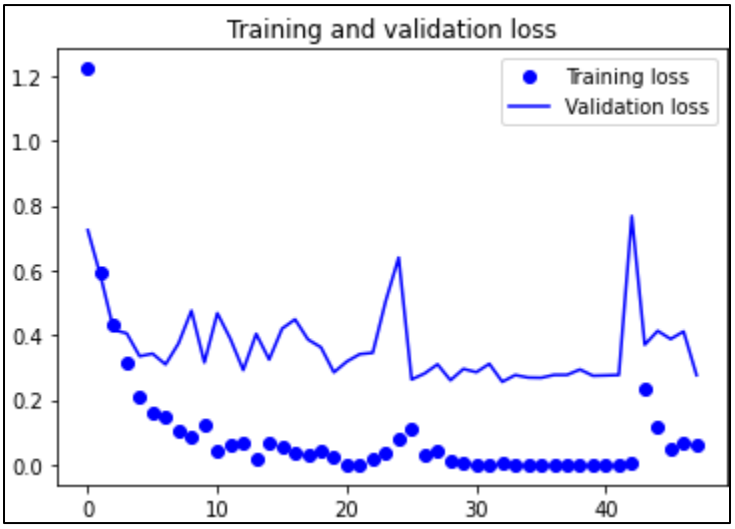


5. A Sample output of the **incorrectly** predicted images

6. Plotting **Training accuracy** with **Validation accuracy**



7. Plotting **Training loss** with **Validation loss**



- ***Output Observation on both Colored Images models***

In this data set, data augmentation **didn't make a difference** in the accuracy results.

In fact, it was less than without the data augmentation. We believe the reason for this is that the data is large enough that it doesn't need data augmentation.

On repeated runs to train the model, almost always they are the **same accuracy or have extremely slight differences.**

Transfer Learning Model (ResNet50)

We ran ResNet50 model on the Random Images Dataset and the Cells Dataset

- ### General Random Images Model (Without Data Augmentation)

Timing of Model

Time	Specs
Maximum Epochs: 10 **Epochs reached:** 10 Each Epoch took around 5 minutes **Total Time to train:** 50 minutes **Size of dataset:** 2293 images Size of each image: 256,256,3 (RGB)	Using a laptop with a GPU-enabled tensorflow/keras **GPU used:** Nvidia GTX 1660Ti **GPU Memory:** 6 GB **Available RAM:** 16 GB

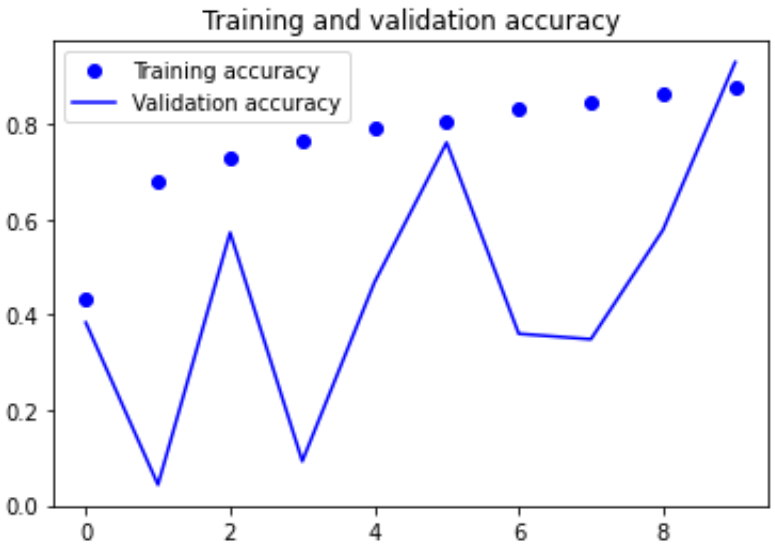
1. Evaluation output: **89%** accuracy and loss equal **0.3**

```
[==============================] - 51s 5s/step - loss: 0.2901 - accuracy: 0.8917
```

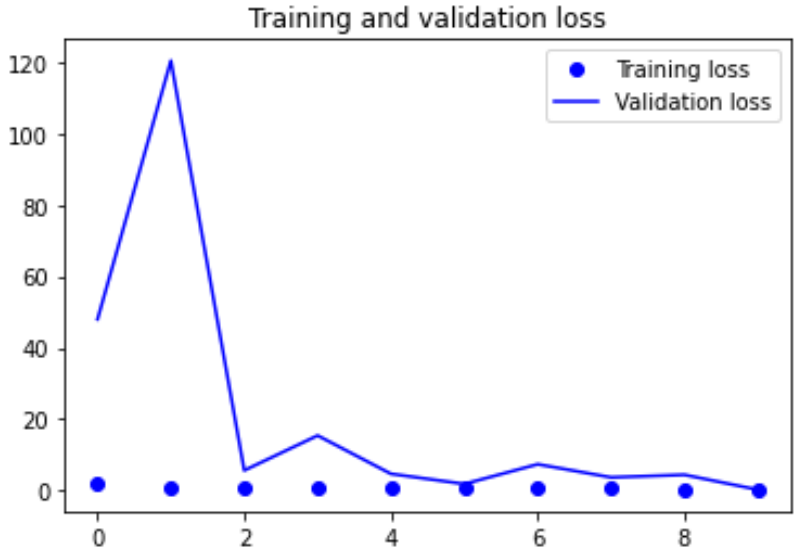
2. **Classification Report**, showing the accuracy equals **94%**

	precision	recall	f1-score	support
Class: Motorbikes	1.00	0.99	1.00	127
Class: helicopter	1.00	1.00	1.00	13
Class: airplanes	1.00	1.00	1.00	117
Class: brain	0.52	1.00	0.69	12
Class: car_side	1.00	1.00	1.00	19
Class: panda	0.00	0.00	0.00	7
Class: chandelier	0.90	1.00	0.95	18
Class: cougar_face	1.00	0.14	0.25	7
Class: crab	0.71	1.00	0.83	5
Class: grand_piano	1.00	0.95	0.97	19
accuracy			0.96	344
macro avg	0.81	0.81	0.77	344
weighted avg	0.95	0.96	0.95	344

3. Plotting **Training accuracy** with **Validation accuracy**



4. Plotting **Training loss** with **Validation loss**



- ## Cells Images Model (Without Data Augmentation)

Timing of Model

Time	Specs
Maximum Epochs: 15 **Epochs reached:** 15 Each Epoch took around 33 seconds **Total Time to train:** 8.3 minutes **Size of dataset:** 862 images **Size of each image:** 382,382,3 (Grayscale , but we make 3 channels for the input for the ResNet50 model, each channel holding the same greyscale values)	Using a laptop with a GPU-enabled tensorflow/keras **GPU used:** Nvidia GTX 1660Ti **GPU Memory:** 6 GB **Available RAM**: 16 GB

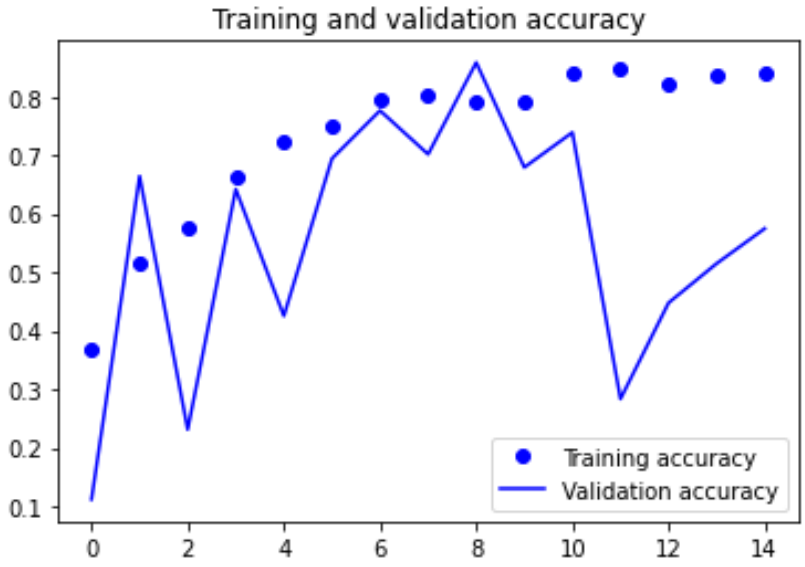
1. Evaluation output: **79%** accuracy and loss equal **2.48**

```
[==============================] - 3s 451ms/step - loss: 2.4879 - accuracy: 0.7939
```

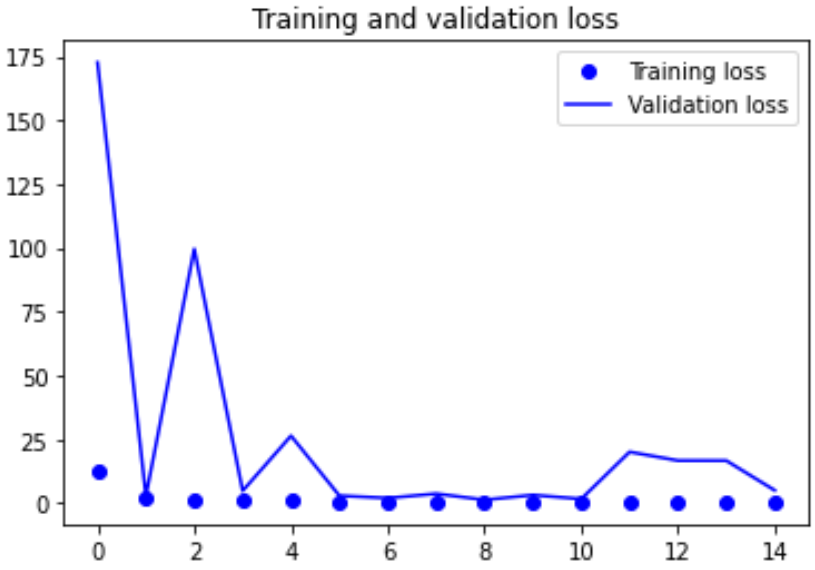
2. **Classification Report**, showing the accuracy equals **79%**

	precision	recall	f1-score	support
Class: actin	0.93	0.88	0.90	16
Class: dna	0.92	1.00	0.96	11
Class: endosome	0.71	0.67	0.69	15
Class: er	0.65	1.00	0.79	13
Class: golgia	0.67	0.92	0.77	13
Class: golgpp	1.00	0.53	0.70	15
Class: lysosome	0.82	0.60	0.69	15
Class: microtubules	0.71	0.62	0.67	8
Class: mitochondria	1.00	0.73	0.84	11
Class: nucleolus	0.78	1.00	0.88	14
accuracy			0.79	131
macro avg	0.82	0.80	0.79	131
weighted avg	0.82	0.79	0.79	131

3. Plotting **Training accuracy** with **Validation accuracy**



4. Plotting **Training loss** with **Validation loss**



- ***Output Observations on ResNet50 Model***

  Using **Resnet50** had a remarkable effect on the cells dataset; the accuracy increased from **73% to 79%**. On the other hand, it didn't increase the accuracy using the random images dataset.

  It was possible to add the **datagen data augmentation** for the **Resnet model** but we believed this would **increase the training time** far too much and it **wouldn't improve the accuracy as much either way.**

  We also wanted our **test comparison** to be based on:
  our model with data augmentation **versus** the transfer learning model with layers practiced on huge numbers of datasets.

Normal from-scratch CNN

- ### Using a simple model

  It uses a simple layer-set which consists of:

  1. **Convolutional layer:**

     Using **valid padding** (which removes 1 from the width and 1 from the height in size)
     Kernel size: 3x3
     Number of filters: 32
     No activation function

  2. **MaxPooling layer:**

     Pool size: 2x2

  3. **Fully connected dense layer:**

     Uses **softmax** activation

  The results of the model after **10 epochs**, took around **17.6 minutes** on the **cell** dataset. Each epoch would take around approximately **2 minutes.**

```
--- Testing the CNN ---
Test Loss: 3.3883665195001673
Test Accuracy: 0.2748091603053435
```

- ***Output Observation on from scratch CNN***

Based on this output alone, we decided to discontinue working on improving the **from scratch model** (It had no room for improvement either way).

As well as the fact that it took this long to train on a small dataset compared to the other dataset for colored images.

And to implement **a from-scratch CNN** that can support multiple layers, we realized that the CNN would have an extremely in-efficient and unoptimized code to run on actual image datasets.

However, it was an interesting experience to see how the actual layers are implemented and what happens inside each of them in terms of code and the idea of trying to write a "**Keras from scratch**" that supports adding multiple layers.

We've also added an optional folder in the GitHub containing the code for the from-scratch CNN using 1 layer-set.