SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30049

ASSIGNMENT 1

BIRD SPECIES CLASSIFICATION

(DUE 20/10/24 - 23:59P.M)

NAME & STUDENTID	:	**Nguyen Gia Binh - 104219428**
CLASS	:	**COS30082 - Applied Machine Learning**
LECTURER	:	**Mr. Duong Trung Tin** (tinduong@swin.edu.au)
TUTOR	:	**Mr. Duong Trung Tin** (tinduong@swin.edu.au)

Hanoi, October 20, 2024

TABLE OF CONTENT

I) Executive Summary

1) Project Overview

This assignment, undertaken by students at Swinburne University of Technology for the COS30082 Applied Machine Learning course, focused on implementing various types of learning models for image classification. The primary goal was to improve classification accuracy and reduce the loss value on the UCSD-200 dataset by utilizing advanced neural network architectures such as ResNet50, MobileNet, and EfficientNet, enhanced with techniques like data augmentation and regularization strategies. The project also aimed to minimize the overfitting issues caused by the dataset.

2) Objective

The main objective is to leverage existing methods, apply them and compare them to see which one solve the overfitting issue best

3) Achievement

- **Learning Strategy application**: Demonstrated that Transfer, Ensemble learning has been effectively applied and well as a hybrid of some existing learning style.
- **Data Augmentation**: Demonstrated that data augmentation has been applied to an extend to help with the overfitting issue.
- **Comparative study**: The study analyze how each model, learning style implemented perform on the given dataset. What did each one does differently to address the overfitting issue that exist in the dataset

II) Introduction

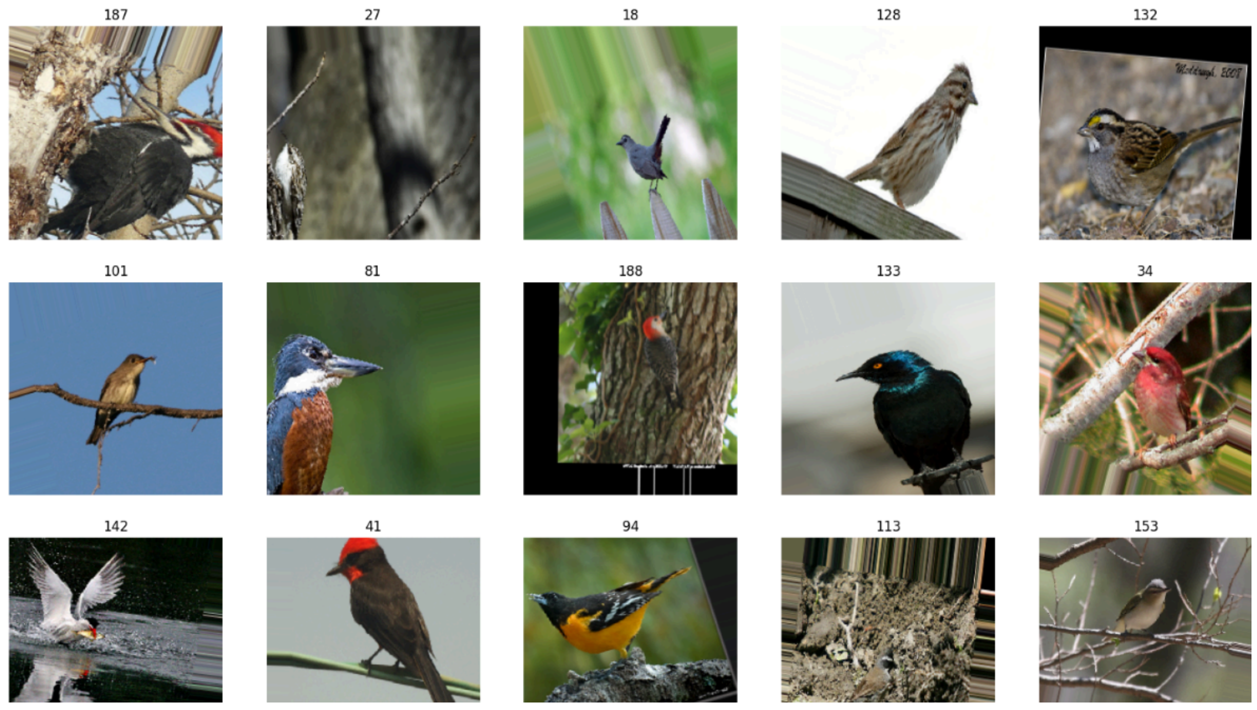
Assignment 1 focuses on bird species classification using the Caltech UCSD 200 Birds dataset. The dataset has a critical flaw: it does not have enough data points, with only 4829 images for training and 1204 for testing, while there are 200 classes to be classified. On average, each class has only around 20 to 30 images, and some classes have fewer than 20 images. As a result, overfitting issues are quite apparent in this task. To address this, various methods have been applied, including different learning strategies, data augmentation, and regularization. The results have varied, but by the end of the assignment, the methods provided some improvement and mitigated the overfitting issue to some extent.

III) Methodology

1) Data processing

a) Data preparation

The process starts by defining the constants for batch size, with BATCH_SIZE = 64. Then, data directories for training and testing, along with their corresponding label files, were established on the Kaggle platform. Next, the train.txt and test.txt files, which contain mappings for images and their classes, were read and parsed into lists before being converted into Pandas dataframes, ensuring each filename was correctly linked to its class label. The dataframes were sorted by the original order provided to maintain consistency throughout the operations. This step is necessary because platforms like Google Colab or Kaggle have their own default sorting methods, so ensuring the files are in their original order is critical for matching images with their labels during model training and evaluation. Finally, I visualized a portion of a batch from the training dataset to inspect the processed images and ensure data integrity (this visualization step was done after data augmentation, so I didn't have to repeat it)

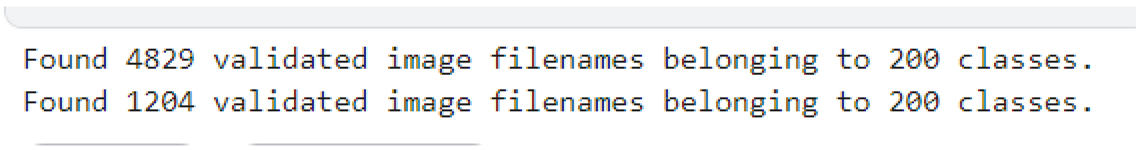


b) Data Augmentation

I utilized the ImageDataGenerator from TensorFlow's Keras library to augment the training dataset, while only rescaling the test dataset to maintain the integrity of the evaluation. Several transformations were applied to simulate different photographic conditions and introduce variability into the training process. Initially, I also used the color shifting method, but I realized that birds are already quite colorful and there is not enough data for each species, so altering the color of the images would be detrimental. Additionally, I tested random erasing, but the dataset images not only contain birds but also have large backgrounds relative to the birds. As a result, the black boxes could not be placed strategically enough to introduce variety and robustness into the training data. After multiple tests, here is the final list of augmentations used:

- **Rescaling**: All images were rescaled by a factor of 1/255.0 to normalize pixel values between 0 and 1.
- **Shear Transformation**: A shear range of 0.2 introduced geometrical transformations to simulate an angled viewing perspective.
- **Zoom Range**: A zoom range of 0.2 to allow the model to learn from various scales of the images.
- **Horizontal Flip**: Images were randomly flipped horizontally..
- **Rotation**: Up to 20 degrees of rotation were put in to account for various orientations of objects within the images.
- **Width and Height Shifts**: Up to 20% of the total width and height shift should helped the model learn to recognize parts of objects and their incomplete representations.

Next, the data generator shuffled the training data to ensure each batch was a random sample, promoting robust learning. The images were then resized to 224x224 pixels, which is the input size suitable for ResNet, MobileNet, and EfficientNet. For the test dataset, I only rescaled the images to ensure the model's performance was evaluated on the original test images. Both datasets were created using the flow_from_dataframe method, which directly linked the images to their labels from the dataframes, streamlining the data preparation process.



```
Found 4829 validated image filenames belonging to 200 classes.
Found 1204 validated image filenames belonging to 200 classes.
```

2) Model

a) Custom Hybrid Learning strategy

In this strategy, I employed transfer learning, specifically using ResNet50 as the backbone for feature extraction, followed by a few additional layers including global pooling, dropout, and Dense layers. I also utilized ReduceLROnPlateau to reduce the learning rate when the validation loss did not decrease for about five epochs.

The core of this strategy lies in how I manage the learning process. After testing the model repeatedly and fine-tuning it over and over, I observed that the model always converged impressively at certain epochs. I purposefully set the training in small increments, running a few epochs at a time, until I could clearly see the onset of overfitting. At that point, I would stop the training and save the model. This process was repeated multiple times, allowing me to select the best seeds before moving on to the next phase.

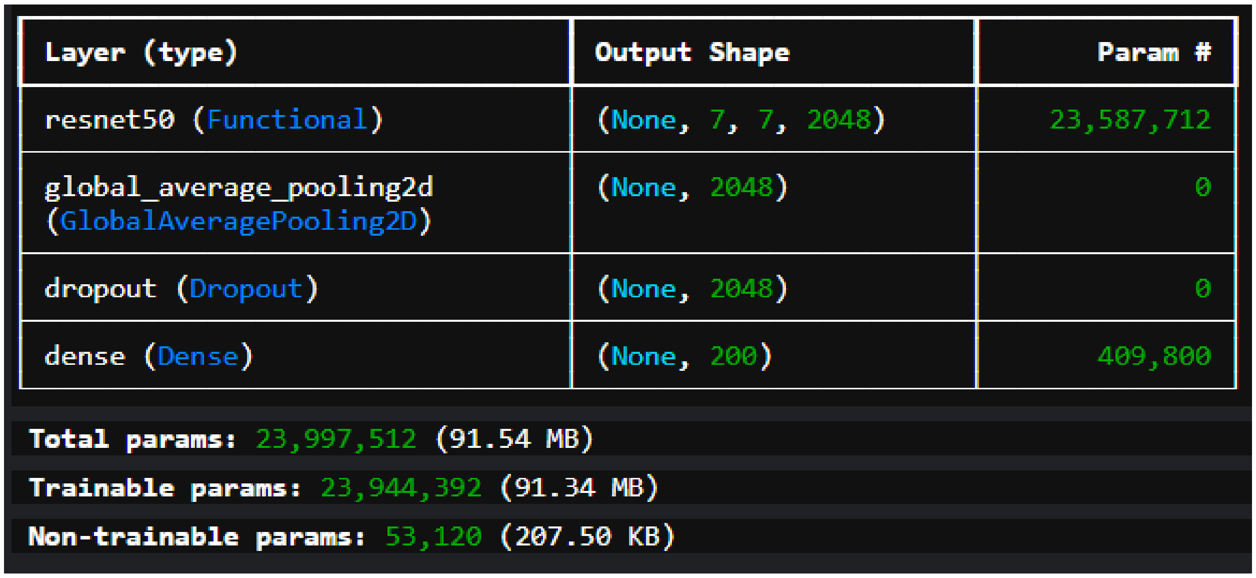
In the subsequent phases, I loaded the best seed model and trained it further, adjusting the learning rate and applying different constraints and regularization techniques such as label smoothing, gradient clipping, and using different learning rate schedulers like cyclic and cosine annealing.

Since this style of training takes a significant amount of time, rather than loading the entire training dataset in each epoch, I adopted a mini-batch approach in the first phase. This involved feeding 76 images per epoch, which took 64 epochs for the model to see the entire training dataset. This approach helped to speed up convergence and introduced some noise into each epoch for better generalization.

The starting phase is the most critical, as it seems to determine whether the model will achieve optimal weights.

- **ResNet50**: pre-trained on ImageNet, with it top layers frozen and only 2 bottom layers are trainable. It take in an input shape of (224, 224, 3).
- **Global Average Pooling 2D**: Condenses feature maps to a single vector per channel.
- **Dropout** (0.25): Mitigates overfitting by randomly disabling 25% of the neurons during training.
- **Dense Output Layer**: 200 units representing 200 classes with softmax activation, L2 regularization at 0.02, and max norm constraint of 3.
- **Compilation**: Optimizer: Adam with a learning rate of 1e-06. Use Categorical crossentropy for loss function and accuracy for metric.

- **ReduceLROnPlateau**: Reduces learning rate by 0.5 if no improvement in val_loss for 4 epochs, verbose is set at 1 for transparency.



```
┌───────────────────────────┬──────────────────────┬──────────────┐
│ Layer (type)              │ Output Shape         │      Param # │
├───────────────────────────┼──────────────────────┼──────────────┤
│ resnet50 (Functional)     │ (None, 7, 7, 2048)   │   23,587,712 │
├───────────────────────────┼──────────────────────┼──────────────┤
│ global_average_pooling2d  │ (None, 2048)         │            0 │
│ (GlobalAveragePooling2D)  │                      │              │
├───────────────────────────┼──────────────────────┼──────────────┤
│ dropout (Dropout)         │ (None, 2048)         │            0 │
├───────────────────────────┼──────────────────────┼──────────────┤
│ dense (Dense)             │ (None, 200)          │      409,800 │
└───────────────────────────┴──────────────────────┴──────────────┘
 Total params: 23,997,512 (91.54 MB)
 Trainable params: 23,944,392 (91.34 MB)
 Non-trainable params: 53,120 (207.50 KB)
```

After the starting phase come phase N, the architecture doesnt change beyond this point, only the parameter value after each training session:

- **Dropout Rate**: Updated to 0.4 for increased regularization
- **Kernel Regularizer**: ElasticNet (L1 and L2) regularization with l1=0.01 and l2=0.03.
- **Compilation**:Adam optimizer with a learning rate of 1e-06 and gradient clipping value of 1.0. CategoricalCrossentropy as loss function with label smoothing at 0.1. Accuracy for metric.
- **ReduceLROnPlateau:** halve the learning rate if no improvement in validation loss is seen over 4 epochs, minimum learning rate set to 5e-07.

b) Ensemble Learning

In this ensemble learning approach, I integrated transfer learning by utilizing three common pre-trained models for image feature extraction: ResNet50, MobileNetV2, and EfficientNetB0. The uniqueness of this strategy lies in the layered integration of these models to enhance the robustness of the feature set. Each model, pre-trained on ImageNet, serves as a distinct pathway for processing input data, ensuring diverse feature extraction.

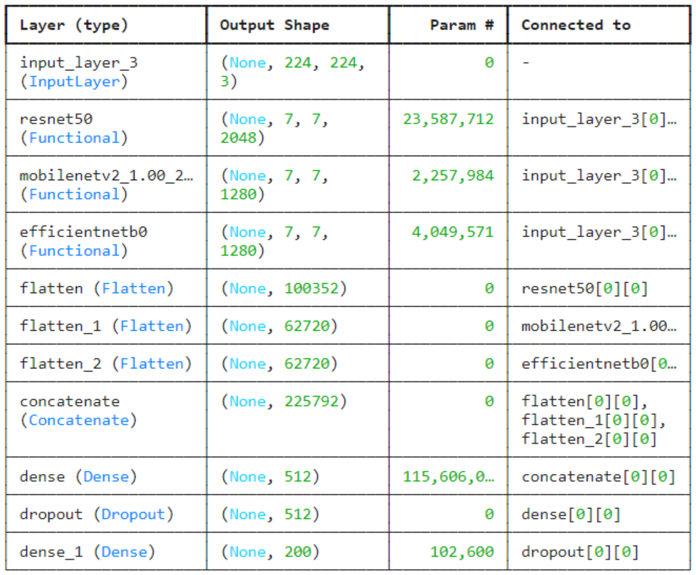
The ensemble method benefits significantly from the parallel use of these architectures, combining their outputs to form a more comprehensive feature vector before classification. This is followed by regularization techniques, including Dropout and L2 regularization, to combat overfitting and ensure the model remains generalizable to unseen data.

Additionally, this strategy dynamically adjusts the learning rate using the ReduceLROnPlateau callback and employs EarlyStopping to halt training at the optimal point, preventing overfitting by monitoring the validation loss.

Unlike the custom hybrid model mentioned earlier, this ensemble model is trained on the full training dataset from the start of every epoch. I aim to evaluate how well the model generalizes when the full range of features is passed through.

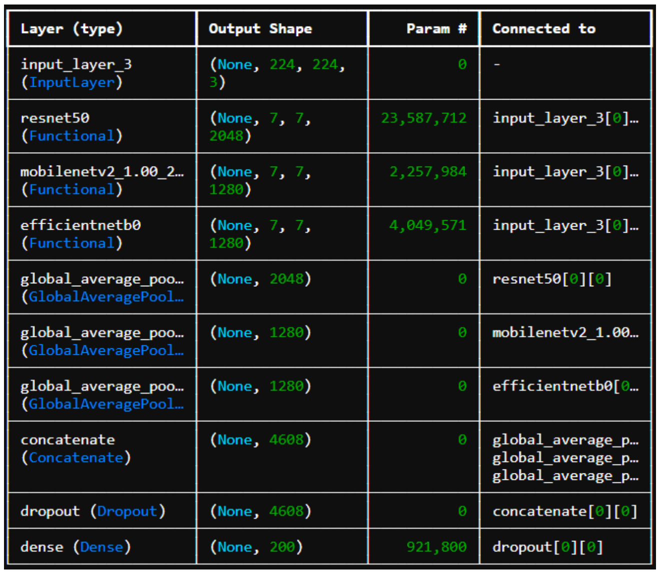
In this strategy, I created three different models to determine which approach would yield the best results. These approaches include concatenating the output of all three models followed by a Flatten layer, concatenating the output followed by a GlobalAveragePooling2D layer, and using a purely soft-voting model. All three models were accompanied by a Dropout layer with a rate of 0.4 for better generalization.

- **Concatenate-Flatten:** as the name implied, after the data pass through ResNet50, MobileNetV2, EfficientNetB0 (only the last 3 layer of these model is trainable) the result of these pre-trained model got Flatten to preserve the spatial element and then concatenated. Finally they got put through 2 Dense layers, one at 512 and the other is 200 as in 200 classes. This is done so that the model can learn more complex features but at the cost of computational power.



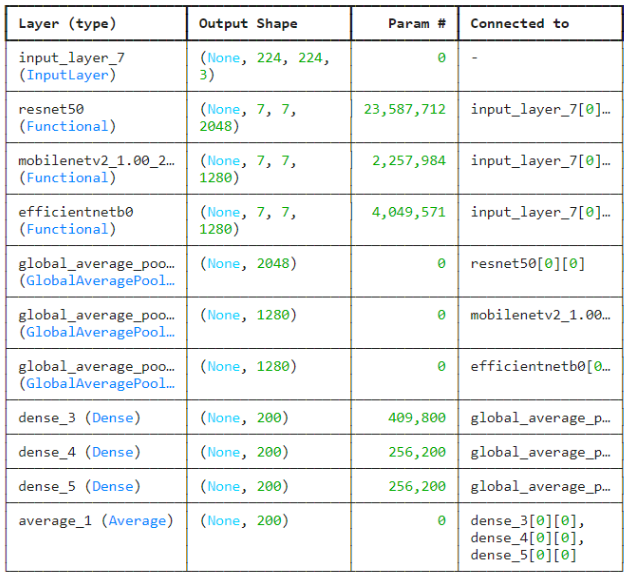
Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0	-
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712	input_layer_3[0]…
mobilenetv2_1.00_2… (Functional)	(None, 7, 7, 1280)	2,257,984	input_layer_3[0]…
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571	input_layer_3[0]…
flatten (Flatten)	(None, 100352)	0	resnet50[0][0]
flatten_1 (Flatten)	(None, 62720)	0	mobilenetv2_1.00…
flatten_2 (Flatten)	(None, 62720)	0	efficientnetb0[0…
concatenate (Concatenate)	(None, 225792)	0	flatten[0][0], flatten_1[0][0], flatten_2[0][0]
dense (Dense)	(None, 512)	115,606,0…	concatenate[0][0]
dropout (Dropout)	(None, 512)	0	dense[0][0]
dense_1 (Dense)	(None, 200)	102,600	dropout[0][0]

- **Concatenate-GlobalAveragePooling2D:** as the name implied, after the data pass through ResNet50, MobileNetV2, EfficientNetB0 (only the last 3 layer of these model is trainable) the result of these pre-trained model go through pooling and then concatenated. Using would make the model lose some of the spatial element but at the same time pooling layer is very common so i want to see if the result would be better.



Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0	-
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712	input_layer_3[0]…
mobilenetv2_1.00_2… (Functional)	(None, 7, 7, 1280)	2,257,984	input_layer_3[0]…
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571	input_layer_3[0]…
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	resnet50[0][0]
global_average_poo… (GlobalAveragePool…	(None, 1280)	0	mobilenetv2_1.00…
global_average_poo… (GlobalAveragePool…	(None, 1280)	0	efficientnetb0[0…
concatenate (Concatenate)	(None, 4608)	0	global_average_p… global_average_p… global_average_p…
dropout (Dropout)	(None, 4608)	0	concatenate[0][0]
dense (Dense)	(None, 200)	921,800	dropout[0][0]

- **Soft voting:** Outputs from each model are processed through individual Dense layers corresponding to that model with softmax activation, L2 regularization and max-norm constraints are applied to manage model complexity and reduce overfitting. These outputs are then averaged, implementing a soft voting mechanism that blends the strengths of each model to hopefully increase the robustness and accuracy of the model overall.



Layer (type)	Output Shape	Param #	Connected to
input_layer_7 (InputLayer)	(None, 224, 224, 3)	0	-
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712	input_layer_7[0]…
mobilenetv2_1.00_2… (Functional)	(None, 7, 7, 1280)	2,257,984	input_layer_7[0]…
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571	input_layer_7[0]…
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	resnet50[0][0]
global_average_poo… (GlobalAveragePool…	(None, 1280)	0	mobilenetv2_1.00…
global_average_poo… (GlobalAveragePool…	(None, 1280)	0	efficientnetb0[0…
dense_3 (Dense)	(None, 200)	409,800	global_average_p…
dense_4 (Dense)	(None, 200)	256,200	global_average_p…
dense_5 (Dense)	(None, 200)	256,200	global_average_p…
average_1 (Average)	(None, 200)	0	dense_3[0][0], dense_4[0][0], dense_5[0][0]

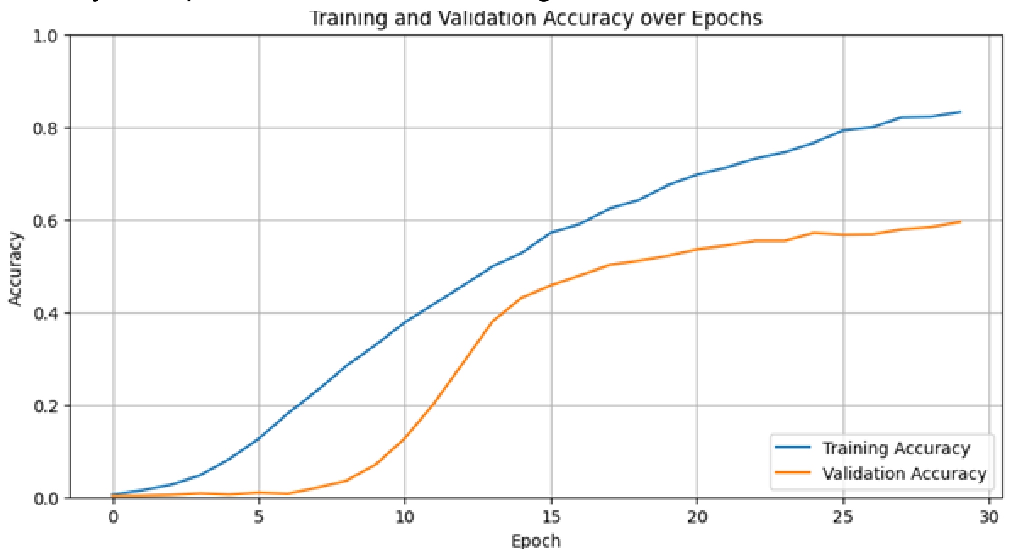
IV) Results and Disusssion

1) Custom Hybrid Learning strategy

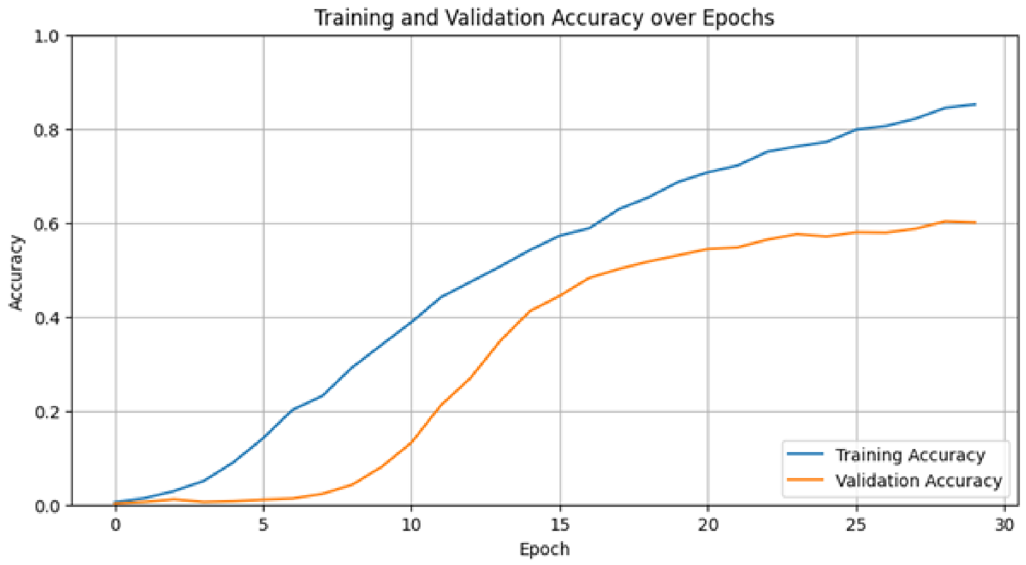
The first few iteration of these approach produce result that are really bad but it give me the insight of where the model is converging well and give me the foundation to follow this way.

- **Phase 1 example**:

This is an image where i first remove batch normalization out of the model due to the model tendency to explode in term of overfitting.



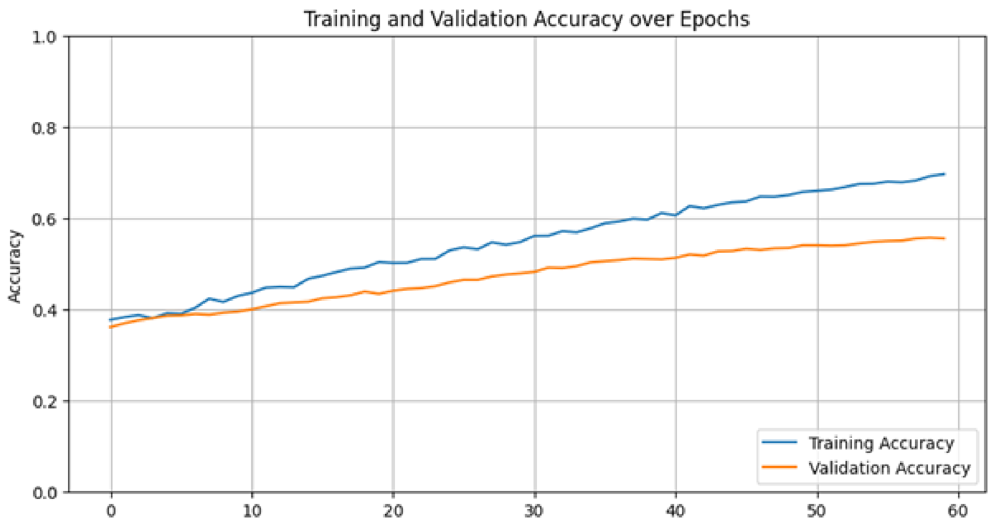
And then when i added max norm to the Dense and dropout layers. Not much change at the end but the speed at which it reach maximum converge is calmer



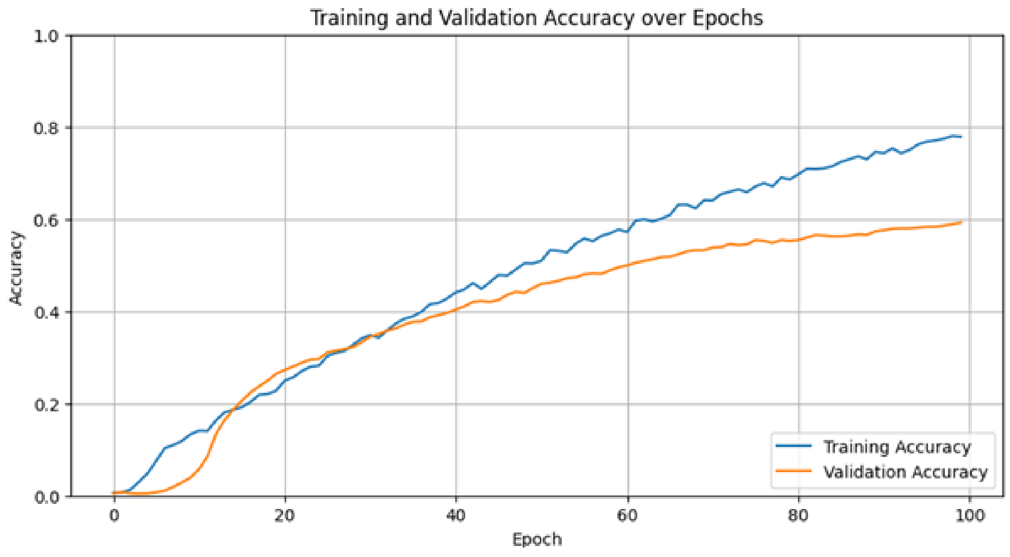
After this i keep experimenting with different value of constraint and learning rate and it seems like slow learning rate (about 1e-06) is giving me the best result

- **Phase N example**:

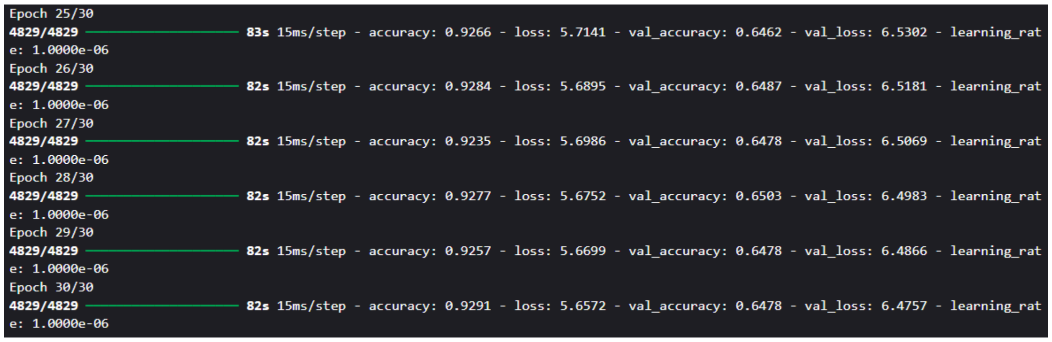
What really made me believe in this hybrid strategy is when i added learning rate scheduler, specifically ReduceLROnPlateau. The training metric and loss suddenly converge at a somewhat stable rate. Overfitting is still a big issue but the graph look way more linear.



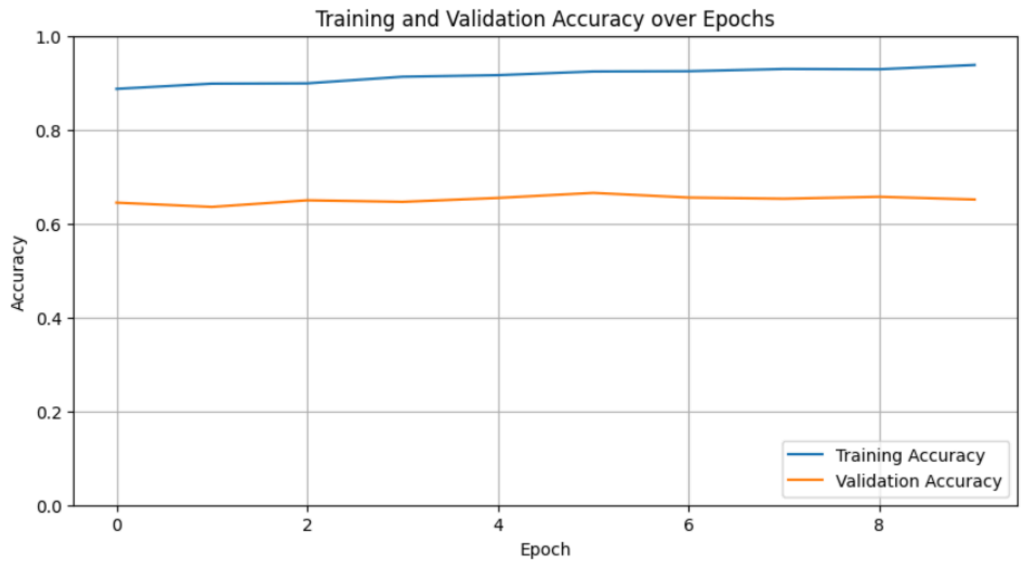
After this i start fine-tuning and found out that dropout value at 0.4 seems to be the limit and if go any further would cause the model to underfit. Furthermore i also found the general threshold that this model can handle before it go into underfitting. The limit is dropout: 0.4, learning rate scheduler: factor <0.2. Here is what it look like when put at the limit

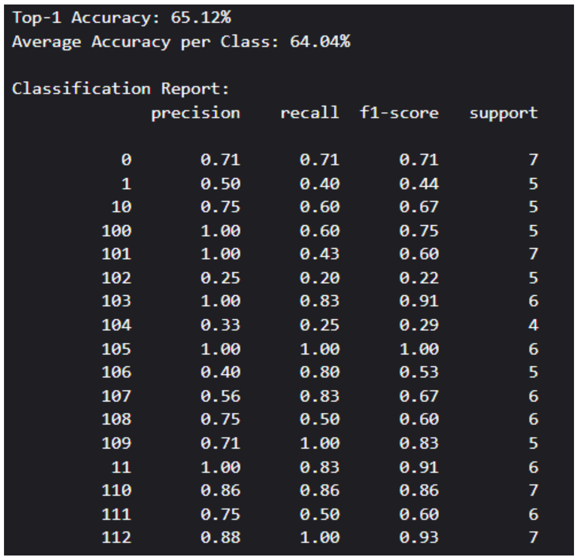


Due to my mistake and bad habit, i didn't check if i had saved the best result and lost it the next day due the Kaggle delete all memory after session end. In the end, the result that I have saved down is quite mediocre. This is the result from yesterday where i test how it will perform on the full training dataset in 1 epochs. This have been re-train back to back for 60 epochs total



```
Epoch 25/30
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 83s 15ms/step - accuracy: 0.9266 - loss: 5.7141 - val_accuracy: 0.6462 - val_loss: 6.5302 - learning_rat
e: 1.0000e-06
Epoch 26/30
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 82s 15ms/step - accuracy: 0.9284 - loss: 5.6895 - val_accuracy: 0.6487 - val_loss: 6.5181 - learning_rat
e: 1.0000e-06
Epoch 27/30
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 82s 15ms/step - accuracy: 0.9235 - loss: 5.6986 - val_accuracy: 0.6478 - val_loss: 6.5069 - learning_rat
e: 1.0000e-06
Epoch 28/30
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 82s 15ms/step - accuracy: 0.9277 - loss: 5.6752 - val_accuracy: 0.6503 - val_loss: 6.4983 - learning_rat
e: 1.0000e-06
Epoch 29/30
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 82s 15ms/step - accuracy: 0.9257 - loss: 5.6699 - val_accuracy: 0.6478 - val_loss: 6.4866 - learning_rat
e: 1.0000e-06
Epoch 30/30
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 82s 15ms/step - accuracy: 0.9291 - loss: 5.6572 - val_accuracy: 0.6478 - val_loss: 6.4757 - learning_rat
e: 1.0000e-06
```





```
Top-1 Accuracy: 65.12%
Average Accuracy per Class: 64.04%

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.71      0.71         7
           1       0.50      0.40      0.44         5
          10       0.75      0.60      0.67         5
         100       1.00      0.60      0.75         5
         101       1.00      0.43      0.60         7
         102       0.25      0.20      0.22         5
         103       1.00      0.83      0.91         6
         104       0.33      0.25      0.29         4
         105       1.00      1.00      1.00         6
         106       0.40      0.80      0.53         5
         107       0.56      0.83      0.67         6
         108       0.75      0.50      0.60         6
         109       0.71      1.00      0.83         5
          11       1.00      0.83      0.91         6
         110       0.86      0.86      0.86         7
         111       0.75      0.50      0.60         6
         112       0.88      1.00      0.93         7
```

Overall, the result is not very good, Average accuracy does match-up with the last few validation loss and the same can be said for accuracy.

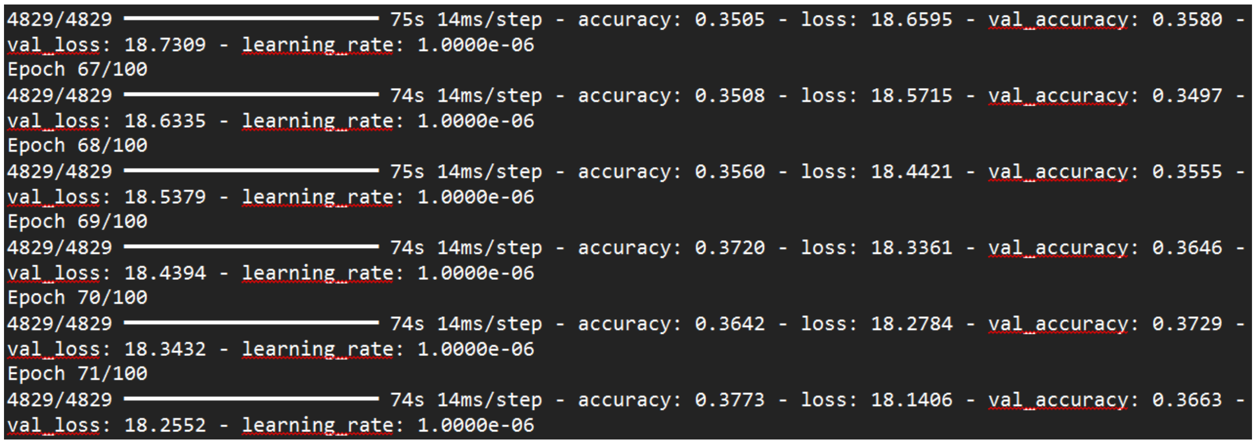
2) Ensemble Learning

First of all, i want to say due to the resources require to run ensemble learning, i couldnt finished training any of the model and here is the reasons:

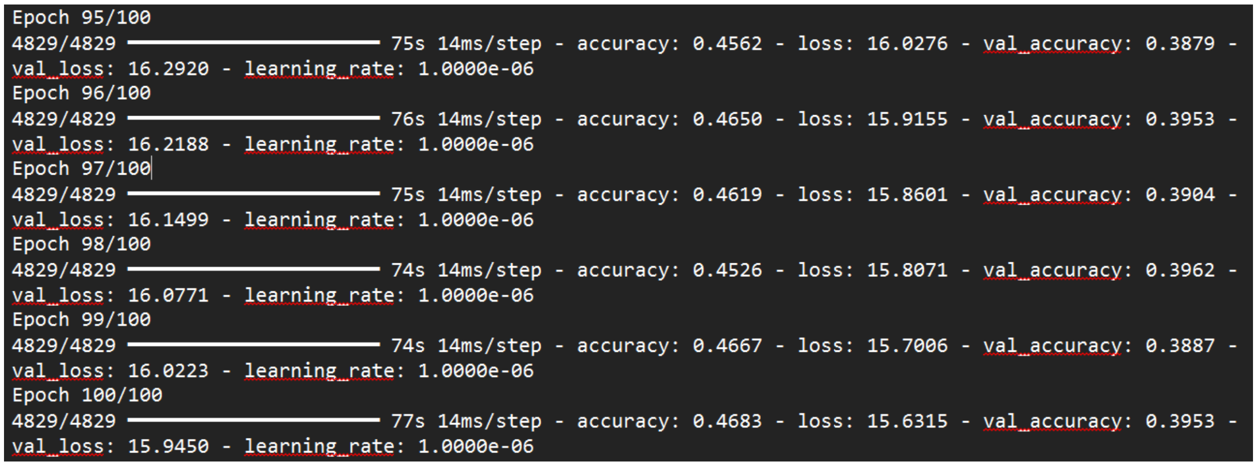
- Kaggle resource: 200 epochs in any of the 3 models would take 3-4 hours to train using the GPU P100 on kaggle, to train any of the model fully, it would take around 600-1000 epochs which would take any where from 12-20 hours (i have tested to 400 epochs/model and it took around 10 hours). Kaggle only offer 30 hours per week for 1 account so it is just impossible for me to run the models to it full potential.
- Kaggle Runtime: Every once in a while i would need to interact with the website to keep kaggle running, so during the day i can but sparingly, at night i just cant and after a few hours, Kaggle automatically turn off the session and delete all data related to that session.
- Kaggle bug: Many times already, kaggle give some bug which would 10x the loss value or even the accuracy even though the model stay the same down to the parameter value. This would force me to re-run the instance to get the correct value.

a) Concatenate-Flatten

These 2 images are in the first 100 epochs when i first tested the model out 2 days ago:



```
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 75s 14ms/step - accuracy: 0.3505 - loss: 18.6595 - val_accuracy: 0.3580 -
val_loss: 18.7309 - learning_rate: 1.0000e-06
Epoch 67/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 74s 14ms/step - accuracy: 0.3508 - loss: 18.5715 - val_accuracy: 0.3497 -
val_loss: 18.6335 - learning_rate: 1.0000e-06
Epoch 68/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 75s 14ms/step - accuracy: 0.3560 - loss: 18.4421 - val_accuracy: 0.3555 -
val_loss: 18.5379 - learning_rate: 1.0000e-06
Epoch 69/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 74s 14ms/step - accuracy: 0.3720 - loss: 18.3361 - val_accuracy: 0.3646 -
val_loss: 18.4394 - learning_rate: 1.0000e-06
Epoch 70/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 74s 14ms/step - accuracy: 0.3642 - loss: 18.2784 - val_accuracy: 0.3729 -
val_loss: 18.3432 - learning_rate: 1.0000e-06
Epoch 71/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 74s 14ms/step - accuracy: 0.3773 - loss: 18.1406 - val_accuracy: 0.3663 -
val_loss: 18.2552 - learning_rate: 1.0000e-06
```

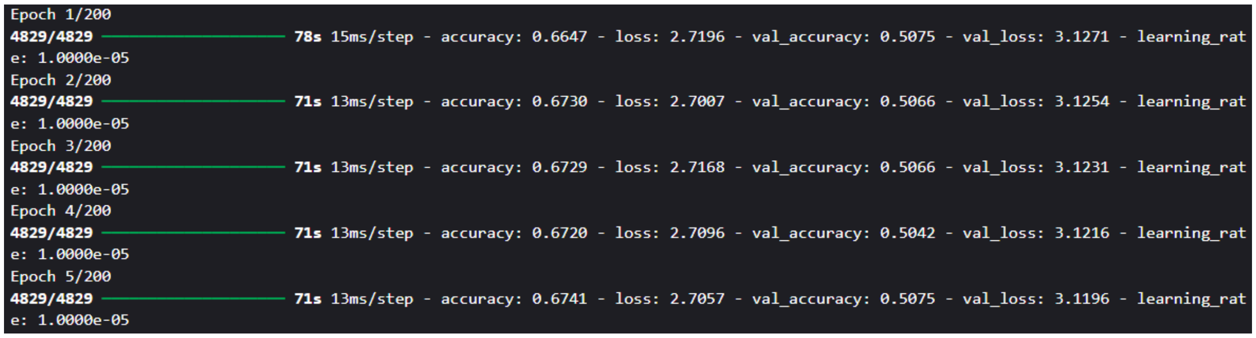


```
Epoch 95/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 75s 14ms/step - accuracy: 0.4562 - loss: 16.0276 - val_accuracy: 0.3879 -
val_loss: 16.2920 - learning_rate: 1.0000e-06
Epoch 96/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 76s 14ms/step - accuracy: 0.4650 - loss: 15.9155 - val_accuracy: 0.3953 -
val_loss: 16.2188 - learning_rate: 1.0000e-06
Epoch 97/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 75s 14ms/step - accuracy: 0.4619 - loss: 15.8601 - val_accuracy: 0.3904 -
val_loss: 16.1499 - learning_rate: 1.0000e-06
Epoch 98/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 74s 14ms/step - accuracy: 0.4526 - loss: 15.8071 - val_accuracy: 0.3962 -
val_loss: 16.0771 - learning_rate: 1.0000e-06
Epoch 99/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 74s 14ms/step - accuracy: 0.4667 - loss: 15.7006 - val_accuracy: 0.3887 -
val_loss: 16.0223 - learning_rate: 1.0000e-06
Epoch 100/100
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 77s 14ms/step - accuracy: 0.4683 - loss: 15.6315 - val_accuracy: 0.3953 -
val_loss: 15.9450 - learning_rate: 1.0000e-06
```

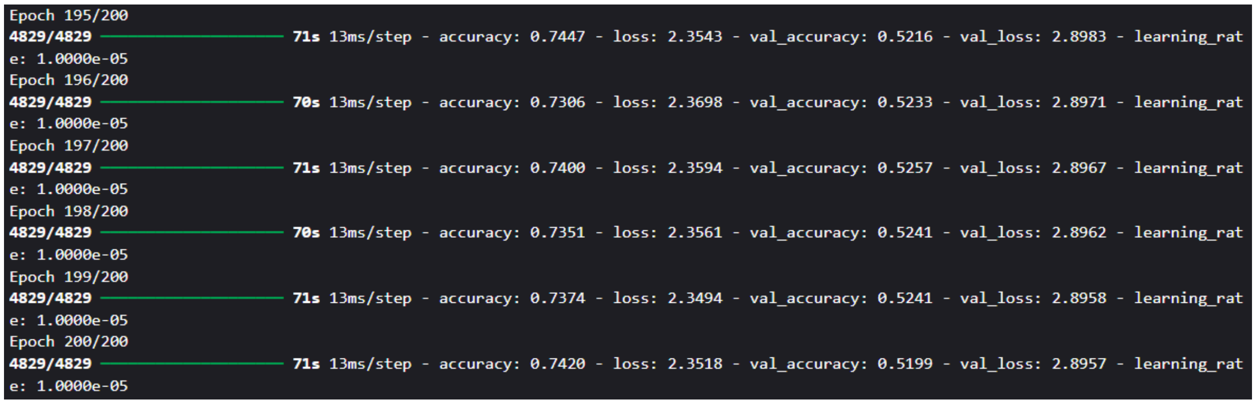
Some testing after this confirm to to that it seems like ensemble learning in this scenario just perform better with learning around 1e-6 -> 5e-6. Looking at the images we can see that the model have little to no overfit in the first 70 epochs, after this the accuracy different between train and test keep widen apart indicating overfit.

b) Concatenate-GlobalAveragePooling2D

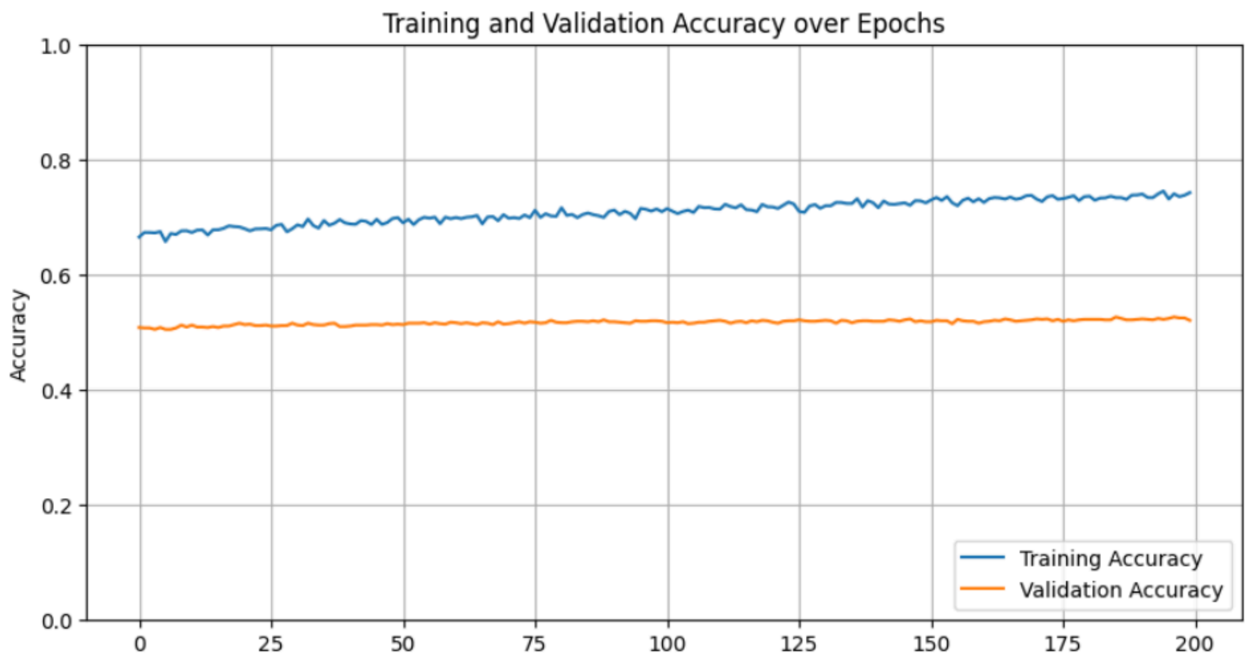
These images is when i finished training 400 epochs for this model



```
Epoch 1/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 78s 15ms/step - accuracy: 0.6647 - loss: 2.7196 - val_accuracy: 0.5075 - val_loss: 3.1271 - learning_rat
e: 1.0000e-05
Epoch 2/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.6730 - loss: 2.7007 - val_accuracy: 0.5066 - val_loss: 3.1254 - learning_rat
e: 1.0000e-05
Epoch 3/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.6729 - loss: 2.7168 - val_accuracy: 0.5066 - val_loss: 3.1231 - learning_rat
e: 1.0000e-05
Epoch 4/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.6720 - loss: 2.7096 - val_accuracy: 0.5042 - val_loss: 3.1216 - learning_rat
e: 1.0000e-05
Epoch 5/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.6741 - loss: 2.7057 - val_accuracy: 0.5075 - val_loss: 3.1196 - learning_rat
e: 1.0000e-05
```



```
Epoch 195/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.7447 - loss: 2.3543 - val_accuracy: 0.5216 - val_loss: 2.8983 - learning_rat
e: 1.0000e-05
Epoch 196/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 70s 13ms/step - accuracy: 0.7306 - loss: 2.3698 - val_accuracy: 0.5233 - val_loss: 2.8971 - learning_rat
e: 1.0000e-05
Epoch 197/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.7400 - loss: 2.3594 - val_accuracy: 0.5257 - val_loss: 2.8967 - learning_rat
e: 1.0000e-05
Epoch 198/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 70s 13ms/step - accuracy: 0.7351 - loss: 2.3561 - val_accuracy: 0.5241 - val_loss: 2.8962 - learning_rat
e: 1.0000e-05
Epoch 199/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.7374 - loss: 2.3494 - val_accuracy: 0.5241 - val_loss: 2.8958 - learning_rat
e: 1.0000e-05
Epoch 200/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - accuracy: 0.7420 - loss: 2.3518 - val_accuracy: 0.5199 - val_loss: 2.8957 - learning_rat
e: 1.0000e-05
```

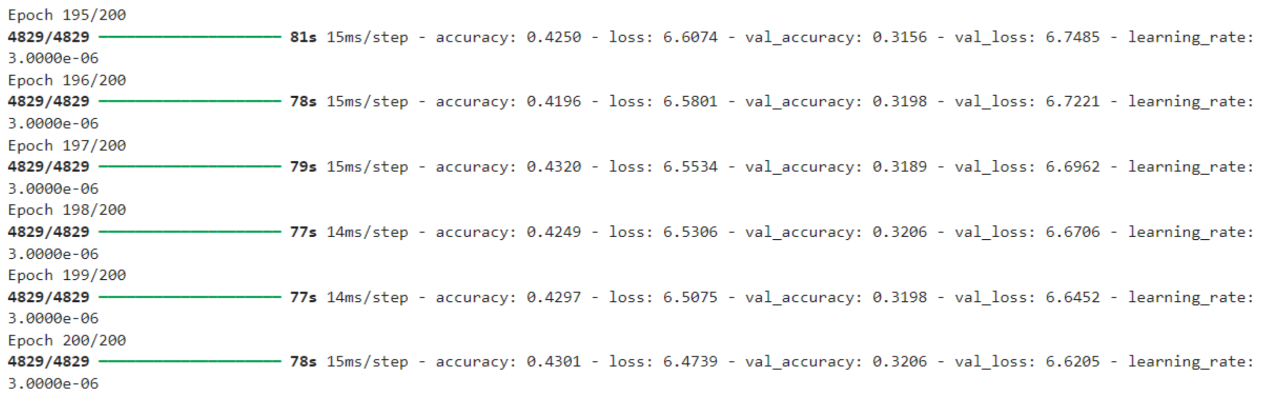


Overall, we can see that the model is learning, just very slowly, slower than expected because before i put 1e-5 as the learning rate, it was 5e-6. Validation loss is actually the lowest in all 4 models tested in this report. Furthermore, we can see that the validation accuracy is stuck at around 52%, at the 201 epoch train accuracy start at 66% but get to

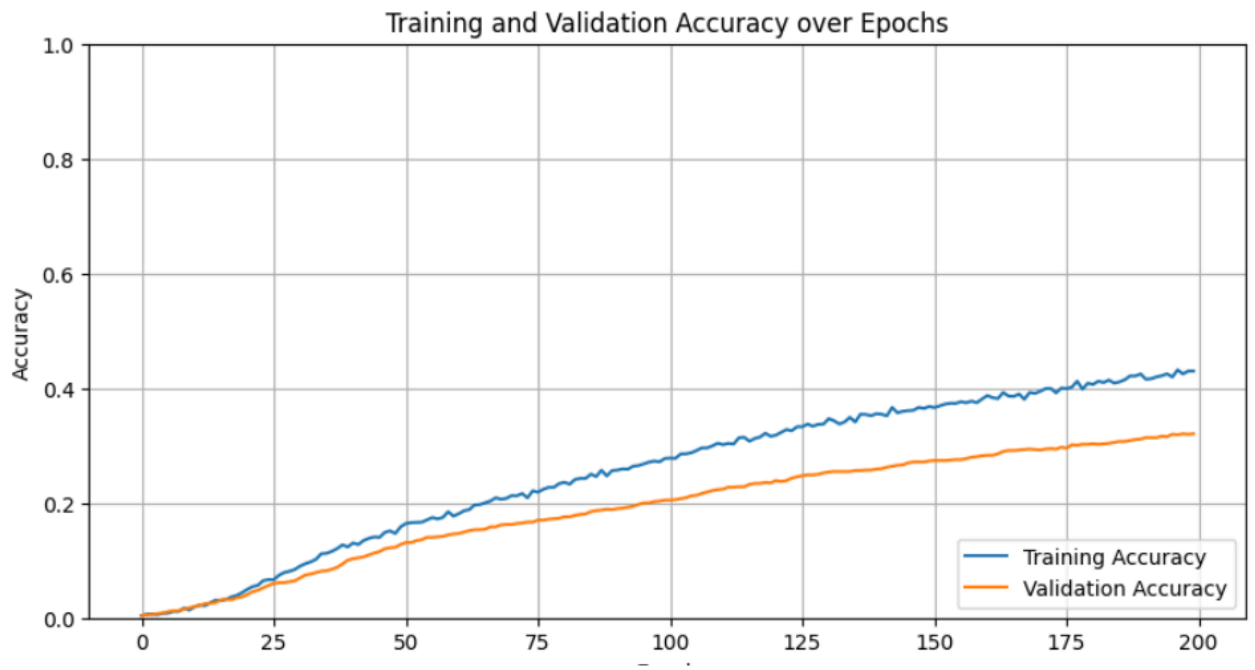
74 at the end of the training session. This indicates the model has reach the plateau with properly no way of escaping unless i switch the learning rate scheduler to increase the learning rate when needed.

c) Soft Voting

These images is the first 200 epochs, my Kaggle have reach it limit this week so i cant continue with the training.



```
Epoch 195/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 81s 15ms/step - accuracy: 0.4250 - loss: 6.6074 - val_accuracy: 0.3156 - val_loss: 6.7485 - learning_rate:
3.0000e-06
Epoch 196/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 78s 15ms/step - accuracy: 0.4196 - loss: 6.5801 - val_accuracy: 0.3198 - val_loss: 6.7221 - learning_rate:
3.0000e-06
Epoch 197/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 79s 15ms/step - accuracy: 0.4320 - loss: 6.5534 - val_accuracy: 0.3189 - val_loss: 6.6962 - learning_rate:
3.0000e-06
Epoch 198/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 77s 14ms/step - accuracy: 0.4249 - loss: 6.5306 - val_accuracy: 0.3206 - val_loss: 6.6706 - learning_rate:
3.0000e-06
Epoch 199/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 77s 14ms/step - accuracy: 0.4297 - loss: 6.5075 - val_accuracy: 0.3198 - val_loss: 6.6452 - learning_rate:
3.0000e-06
Epoch 200/200
4829/4829 ━━━━━━━━━━━━━━━━━━━━ 78s 15ms/step - accuracy: 0.4301 - loss: 6.4739 - val_accuracy: 0.3206 - val_loss: 6.6205 - learning_rate:
3.0000e-06
```



First of all, this model seems to exert more overfitting from early on than the other 2 ensemble models. From the graph it is clear that the model is learning, though overfitting is an issue very early on, the rate at which it increase is very stable and wouldnt explode like other approach.

V) Appendix

1) Acknowledgement

I want to say thank you to the github user name Limteckping45 as i got the code for sorting the label according to the image from him/her. This code is available at this link: https://github.com/Limteckping45/Cos30082_Assignment/blob/main/Bird_Classifier_Training_Model.ipynb

I also want to say thank you to my senior- Mr.Huy Hoang for recommending testing out the ensemble methods to see how it perform. Though the experiment didnt go well, i've got valuable insight into this technique.

2) Dataset

The dataset i used is still CUB-200 but it is from someone on Kaggle and it is available at this link: https://www.kaggle.com/datasets/shmaker/ucsd-200

3) Kaggle notebook

All of my code are run on Kaggle because i dont have a GPU and google colab charge $9.99 to use the GPU P100 while kaggle offer it for free for 30 hours/week. These notebooks are on 3 different accounts because there is just no way 30 hours would be enough to test everything out and fine-tuning the model. Here is the list:

- Custom Hybrid Learning strategy: https://www.kaggle.com/code/nguyentuanducfswhn/noob-custom-hybrid-learning-for-cub-200
- Ensemble Flatten: https://www.kaggle.com/code/binhswinburnehn/asm-1-ensemble-flatten
- Ensemble Global Average Pooling 2D: https://www.kaggle.com/code/nguyentuanducfswhn/asm-1-ensemble-global-average-pooling-2d
- Ensemble Soft Voting: https://www.kaggle.com/code/sntrnmnh/asm-1-soft-voting-incomplete