Assignment Report: Multi-Class Image Classification

Fullname: Dang Khoa Le
Student ID: 103844421
Tutorial: Monday 6.30-8.30 PM

1 Methodology

This project focuses on developing and evaluating deep learning models, in consist of:
+ Base CNN model built from scratch
+ CNN model with dropouts and augmentation
+ ResNet50 transfer learning model

These models are deployed for multi-class image classification task using dataset **"Deep Learning Practice - Image Classification"** from Kaggle, where each class contains 1000 images (total of 10,000). The dataset includes 10 distinct image classes, and the model must generalize to classify unseen samples.

1.1 Dataset and Preprocessing

The dataset, stored in Google Drive under the train/ directory, contains 10 subfolders—each representing a distinct class—with 1,000 images per class. Since the dataset does not include a separate testing set, I manually split the data into training and validation sets using an 80/20 ratio while preserving class distribution through stratified sampling:

```
train_paths, val_paths, train_labels, val_labels = train_test_split(
    file_paths, labels, test_size=0.2, stratify=labels, random_state=SEED)
```

Before feeding images into the models training sessions, several preprocessing steps were applied using TensorFlow's ImageDataGenerator. These include resizing, scaling, augmentation, and batching, ensuring consistency and variability during training.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Label Extraction and Path Construction

- Each image was mapped to its corresponding class name based on its folder.
- This generated a list of file paths (file_paths) and associated class labels (labels).

```
for class_index, class_name in enumerate(class_names):
    class_folder = os.path.join(train_path, class_name)
    for fname in os.listdir(class_folder):
        file_paths.append(os.path.join(class_folder, fname))
        labels.append(class_name)
```

1.1.1. CNN Models' Processing

Input Size: 128x128.
Normalization:

- All pixel values were rescaled from [0, 255] to [0, 1] by dividing by 255.
- Normalization helps the model converge faster during training by standardizing the input range.

Data Augmentation (Training Set Only): To reduce overfitting and improve generalization, the training data was augmented with the following techniques:

- **Horizontal Flipping**: Randomly flips images left-to-right.
- **Rotation**: Rotates images randomly up to 20 degrees.
- **Zooming**: Applies random zoom with a range of 20%.

```
train_gen = ImageDataGenerator(rescale=1./255, horizontal_flip=True, rotation_range=20,
zoom_range=0.2)
```

1.1.2. ResNet50 Model Processing

Input Size: 224×224

Normalization: Used keras.applications.resnet50.preprocess_input to match ImageNet expectations.

```
from tensorflow.keras.applications.resnet50 import preprocess_input
r50_val_gen = ImageDataGenerator(preprocessing_function=preprocess_input)
r50_train_gen = ImageDataGenerator(preprocessing_function=preprocess_input)
```

Data Augmentations: Not applied initially to keep baseline consistent.

1.2 Model Architectures

I implemented and evaluated two CNN architectures:

Model 1: Baseline CNN

- 2 Convolutional layers with ReLU
- 2 MaxPooling layers
- Flatten → Dense(128) → Dense(output)
- Output: Softmax for 10 classes

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(*IMG_SIZE, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Model 2: CNN + Dropout + Augmentation

- Similar to Model 1, but includes:
  - **Dropout(0.3)** after second conv block
  - **Dropout(0.5)** before final dense layer
- Data augmentation improves robustness
- Dropout mitigates overfitting
- Also applying data augmentation

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(*IMG_SIZE, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Model 3: ResNet50 + Transfer Learning

- Architecture:
  - Frozen base: *ResNet50(include_top=False, weights='imagenet')*
  - Custom classifier head:
    - GlobalAveragePooling2D
    - Dense(128, relu)
    - Dropout(0.5)
    - Dense(10, softmax)
- Optimizer: Adam (learning rate = 1e-4)

```
def build_resnet50_transfer():
    r50_model = ResNet50(weights='imagenet', include_top=False, input_shape=(*IMG_SIZE_R50,
3))
    r50_model.trainable = False  # Freeze base
    x = r50_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=r50_model.input, outputs=predictions)
    model.compile(optimizer=Adam(learning_rate=1e-4), loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
```

1.3 Training Details

- **Loss**: categorical_crossentropy (for multi-class)
- **Optimizer**: adam
- **Epochs**: 20
- **Batch Size**: 32
- **Validation**: 20% of val_data splitted

```
history_base = model_base.fit(train_data, epochs=20, validation_data=val_data)
history_augmentation = model_augmentation.fit(train_data, epochs=20, validation_data=val_data)
```

1.4 Model Shapes

This script was executed to visualize and explain the 3 models' shape using summary() function:

```
model_base.summary()
model_augmentation.summary()
model_resnet50.summary()
```

Output was shown on console:

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 158, 158, 32)	896
max_pooling2d_10 (MaxPooling2D)	(None, 79, 79, 32)	0
conv2d_11 (Conv2D)	(None, 77, 77, 64)	18,496
max_pooling2d_11 (MaxPooling2D)	(None, 38, 38, 64)	0
flatten_5 (Flatten)	(None, 92416)	0
dense_14 (Dense)	(None, 128)	11,829,376
dense_15 (Dense)	(None, 10)	1,290

Total params: 11,850,058 (45.20 MB)

Trainable params: 11,850,058 (45.20 MB)

Non-trainable params: 0 (0.00 B)

Table 1. CNN Baseline Model Shape

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 158, 158, 32)	896
max_pooling2d_12 (MaxPooling2D)	(None, 79, 79, 32)	0
conv2d_13 (Conv2D)	(None, 77, 77, 64)	18,496
max_pooling2d_13 (MaxPooling2D)	(None, 38, 38, 64)	0
dropout_8 (Dropout)	(None, 38, 38, 64)	0
flatten_6 (Flatten)	(None, 92416)	0
dense_16 (Dense)	(None, 256)	23,658,752
dropout_9 (Dropout)	(None, 256)	0
dense_17 (Dense)	(None, 10)	2,570

Total params: 23,680,714 (90.33 MB)

Trainable params: 23,680,714 (90.33 MB)

Non-trainable params: 0 (0.00 B)

Table 2. Dropout + Augmentation Model Shape

1.4.1. CNN Baseline Model

Structure Overview:

- Conv2D → MaxPooling2D → Conv2D → MaxPooling2D → Flatten → Dense(128) → Dense(10)
- Total Parameters: **~11.850 million**

Detailed Layer Flow:

- Conv2D(32 filters, 3x3 kernel): The first convolution layer extracts low-level features like edges and textures from input images (size reduces slightly).
- MaxPooling2D(2x2): Down-samples the feature maps to reduce dimensionality and computation.
- Conv2D(64 filters, 3x3 kernel): Learns more complex features by combining basic patterns.
- MaxPooling2D(2x2): Further down-samples feature maps.
- Flatten(): Flattens the 3D output to a 1D vector for the fully connected layers.
- Dense(128): A fully connected layer with 128 neurons for feature interpretation.
- Dense(10): Output layer predicting probabilities for 10 classes (SoftMax activation).

1.4.2. CNN with Dropout and Augmentation Model

Structure Overview:

- Conv2D → MaxPooling2D → Conv2D → MaxPooling2D → Dropout → Flatten → Dense(256) → Dropout → Dense(10)
- Total Parameters: **~23.681 million**

Detailed Layer Flow:

- Similar to the baseline, but with the following additions:
- Dropout(0.3 after conv layers):
  - Introduces random neuron deactivation during training.
  - Reduces the risk of overfitting.
- Dense(256):
  - Increased number of neurons to **learn more complex patterns**.
- Dropout(0.5 after Dense):
  - Further regularizes the model.

1.4.3. Transfer Learning with ResNet50 Model

Structure Overview:

- Pretrained ResNet50 Base (Frozen) → GlobalAveragePooling → Dropout(0.5) → Dense(128) → Dropout(0.3) → Dense(10)
- Total Parameters: **~23.851 million**
  - Trainable params: **~263K** (only the top layers)
  - Non-trainable params: **~23.858M** (ResNet50 frozen)

Detailed Layer Flow:

- ResNet50 base (Frozen):
  - Pretrained on ImageNet.
  - Extracts highly generalized image features such as textures, patterns, object parts.
  - No training on lower layers; only new layers are trained.
- GlobalAveragePooling2D:
  - Replaces the dense flattening operation to reduce parameters.
  - Reduces the tensor to (None, 2048) by averaging spatial dimensions.
- Dropout(0.5):
  - Reduces overfitting.
- Dense(128, ReLU):
  - Fully connected layer for feature compression and discrimination.
- Dropout(0.3):
  - Extra regularization.
- Dense(10, Softmax):
  - Output layer mapping to the 10 classes.

Model	Key Characteristics	Purpose
CNN Baseline	Basic ConvNet, no regularization	Provide a reference benchmark
CNN + Dropout/Augmentation	Added Dropout and larger Dense layer to improve robustness	Mitigate overfitting and enhance learning
ResNet50 Transfer Learning	Pretrained deep network (ResNet50), frozen weights	Leverage large-scale ImageNet features for higher accuracy and faster convergence

Table 3. Models Shape Summary and Reason of Selection

2. Results and Discussion

2.1 Model per Class Accuracy

2.1.1 Methods

To ensure that the predicted labels align correctly with the ground truth across batch-wise inference, I implemented a synchronized evaluation loop. This custom function extracts predictions and true labels in the exact order of processing:

```
def get_y_true_and_pred(model, generator):
    y_true_all, y_pred_all = [], []
    for x_batch, y_batch in generator:
        y_true_all.extend(np.argmax(y_batch, axis=1))
        y_pred_batch = model.predict(x_batch, verbose=0)
        y_pred_all.extend(np.argmax(y_pred_batch, axis=1))
        # Stop after full epoch
        if len(y_true_all) >= generator.samples:
            break
    return np.array(y_true_all), np.array(y_pred_all)
```

This method was applied for all models, ensuring accurate comparison:

- val_data was used for both CNN models,
- r50_val_data was used for ResNet50 with transfer learning.

2.1.2 Results

The classification report (precision, recall, and F1-score per class) reveals how each model performs across individual categories. This deeper evaluation is critical to understanding not just overall performance, but also how well the model handles each class — especially important in imbalanced or multi-class datasets like ours.

To generate this breakdown, I used a utility function:

```
def per_class_accuracy(y_true, y_pred, class_names, model_name="Model"):
    report = classification_report(y_true, y_pred, labels=range(len(class_names)),
target_names=class_names, output_dict=True, zero_division=0)
    # Print the classification report for the current model
    print(classification_report(y_true, y_pred, target_names=class_names, digits=4))
    print(f"\nPer-Class Accuracy (Recall) for {model_name}:")
    for cls in class_names:
        acc = report[cls]["recall"]
        print(f"{cls:10s}: {acc:.4f}")
```

Which:

- Generates a full classification report using scikit-learn's classification_report(),
- Extracts and prints the recall (i.e., per-class accuracy) for each class summary.
- Ensures consistent class alignment by explicitly setting the labels and target_names,
- Handles edge cases where a class might not be predicted at all by setting zero_division=0.

a) Baseline CNN:

- Best performance: *Plantae* (recall: 0.69), *Mammalia* (0.48), *Aves* (0.49), *Arachnida* (0.41)
- Poor performance: *Reptilia* (0.21), *Mollusca* (0.19), *Amphibia* (0.20)
- Insight: The model relies heavily on simpler texture or structure-based cues, performing better on classes with distinct features but struggling with overlapping features or limited samples.

	precision	recall	f1-score	support
Amphibia	0.25	0.20	0.22	200
Animalia	0.35	0.23	0.28	200
Arachnida	0.45	0.41	0.43	202
Aves	0.40	0.49	0.44	200
Fungi	0.42	0.34	0.37	200
Insecta	0.39	0.31	0.34	200
Mammalia	0.29	0.48	0.36	200
Mollusca	0.34	0.19	0.24	200
Plantae	0.37	0.69	0.48	200
Reptilia	0.30	0.21	0.25	200
accuracy			0.36	2002
macro avg	0.36	0.36	0.34	2002
weighted avg	0.36	0.36	0.34	2002

Table 4. Baseline CNN model per-class Accuracy

CNN + Dropout and Augmentation:

- Improved generalization in some categories like *Fungi* (recall 0.46) and *Arachnida* (0.51), but worsened in *Amphibia* (0.13) and *Mollusca* (0.12).
- This suggests that while regularization helped reduce overfitting (see section 2.4), it may have introduced underfitting or noise sensitivity in finer-grained classes.

	precision	recall	f1-score	support
Amphibia	0.27	0.13	0.18	200
Animalia	0.27	0.14	0.18	200
Arachnida	0.37	0.51	0.43	202
Aves	0.45	0.31	0.37	200
Fungi	0.40	0.46	0.42	200
Insecta	0.32	0.28	0.30	200
Mammalia	0.30	0.48	0.37	200
Mollusca	0.38	0.12	0.18	200
Plantae	0.37	0.60	0.46	200
Reptilia	0.25	0.34	0.29	200
accuracy			0.34	2002
macro avg	0.34	0.34	0.32	2002
weighted avg	0.34	0.34	0.32	2002

Table 5. Dropout and Augmentation CNN model per-class Accuracy

ResNet50 Transfer Learning:

- Strong and consistent recall across all classes, notably:
  - *Fungi*: 0.88
  - *Aves*: 0.86
  - *Plantae*: 0.84
  - *Arachnida*: 0.79
- Observation: No class fell below 68% recall, indicating the robustness of pre-trained features.
- Conclusion: Transfer learning significantly enhances the model's ability to generalize across diverse biological categories, benefiting especially from ImageNet pretraining on real-world images.

	precision	recall	f1-score	support
Amphibia	0.74	0.69	0.71	200
Animalia	0.71	0.74	0.73	200
Arachnida	0.86	0.79	0.82	202
Aves	0.80	0.86	0.83	200
Fungi	0.80	0.88	0.83	200
Insecta	0.81	0.72	0.77	200
Mammalia	0.81	0.76	0.78	200
Mollusca	0.71	0.68	0.70	200
Plantae	0.69	0.84	0.76	200
Reptilia	0.76	0.71	0.74	200
accuracy			0.77	2002
macro avg	0.77	0.77	0.77	2002
weighted avg	0.77	0.77	0.77	2002

Table 6. ResNet50 Transfer Learning model per-class Accuracy

2.1.3 Confusion Matrix

Figures 1-3 display the confusion matrices of all three models, using confusion_matrix function from sklearn and matplotlib.pyplot function, providing a visual representation of misclassification patterns:

- **Baseline CNN** (Figure 1):
  - Exhibits heavy confusion between *Fungi*, *Insecta*, and *Plantae*, indicating shared visual traits.
  - *Mollusca* and *Amphibia* are commonly misclassified into multiple classes.
- **CNN + Dropout and Augmentation** (Figure 2):
  - Shows slightly more dispersed predictions, likely due to regularization effects.
  - Improvements in class-specific predictions (*Arachnida*, *Reptilia*) are visible, but some classes became noisier (*Animalia* scattered into 6+ classes).
- **ResNet50 Transfer Learning** (Figure 3):
  - The matrix exhibits strong diagonal dominance, showing accurate classification.
  - Few off-diagonal entries suggest confident and well-calibrated predictions.
  - Notably, *Plantae* (168/200), *Aves* (172/200), and *Fungi* (175/200) were predicted with very high certainty.

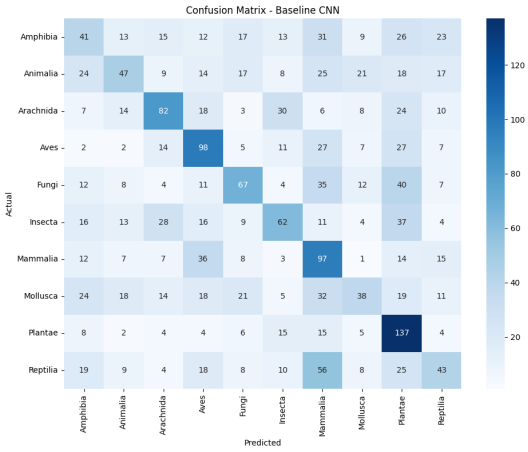

Figure 1

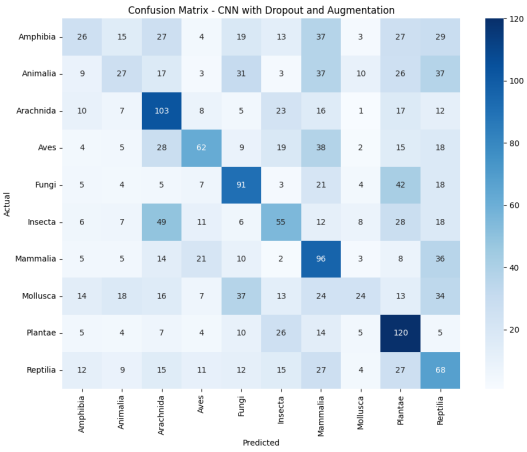

Figure 2

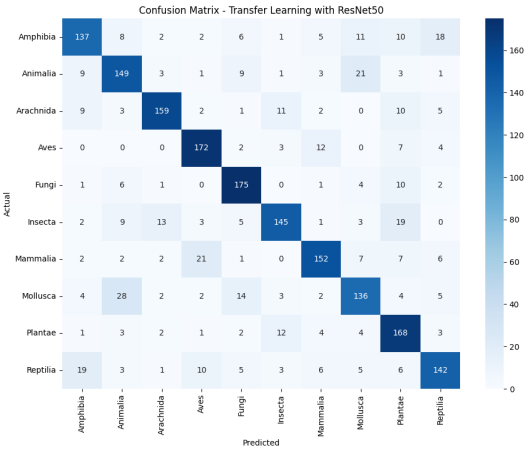

Figure 3

2.2 Metrics

This report two key metrics:

- **Top-1 Accuracy:** Measures how often model's top predicted class matches the ground truth.

```
ybase_pred_probs = model_base.predict(val_data)
ybase_pred = np.argmax(ybase_pred_probs, axis=1)
```

- **Average Accuracy per Class:** Calculates the accuracy for each class independently, then averages these values, ensuring fair performance assessment across class imbalance.

For each class `C`, recall is:

$$\text{Recall}_C = \frac{\text{True Positives}_C}{\text{True Positives}_C + \text{False Negatives}_C}$$

Figure 4. Average per Class Algorithm

```
yaug_pred_probs = model_augmentation.predict(val_data)
yaug_pred = np.argmax(yaug_pred_probs, axis=1)
```

2.3 Evaluation Results

```
Top-1 Accuracy for Base Model: 0.3556
Top-1 Accuracy for Augmented Model: 0.3357
Top-1 Accuracy for ResNet50 Transfer Model: 0.7667
Average Accuracy per Class for Base Model: 0.3556
Average Accuracy per Class for Augmented Model: 0.3355
Average Accuracy per Class for ResNet50 Transferred Model: 0.7667
```

Figure 5. Models Evaluation Top-1 and Average Accuracies from console print

Model	Top-1 Accuracy	Avg Accuracy per Class
Baseline CNN	0.3556	0.3556
CNN + Dropout + Augment	0.3357	0.3355
ResNet50 Transfer Learning	0.7667	0.7667

Table 7. Models Accuracies Comparison table

Interpretation:

- While I expected the dropout and augmentation model to perform better due to regularization, in this particular training run, the baseline model slightly outperformed it in both metrics. This indicates or suggests that:
  - The added complexity and regularization may have slowed convergence or underfitted the training data at this stage.
  - Further tuning (e.g., more epochs, better augmentation balance, optimizer adjustments) may be needed to realize its full potential.
- The **ResNet50 model** significantly outperformed both traditional **CNN** architectures, which states:
  - Transfer learning allowed the model to generalize better despite limited training data by reusing features learned from large-scale ImageNet data.
  - The consistent gap between the ResNet50 and the CNNs confirms the value of transfer learning for small-to-medium datasets.

2.4 Training Curves

Training and validation accuracy curves over 20 epochs reveal the following:

Baseline CNN Model Curve

- **Observation**: The training accuracy appears to increase steadily over epochs, while the validation accuracy either plateaus or grows much more slowly.
- **Interpretation**: This divergence between training and validation indicates overfitting. The model is memorizing training examples but not generalizing as effectively to unseen data.

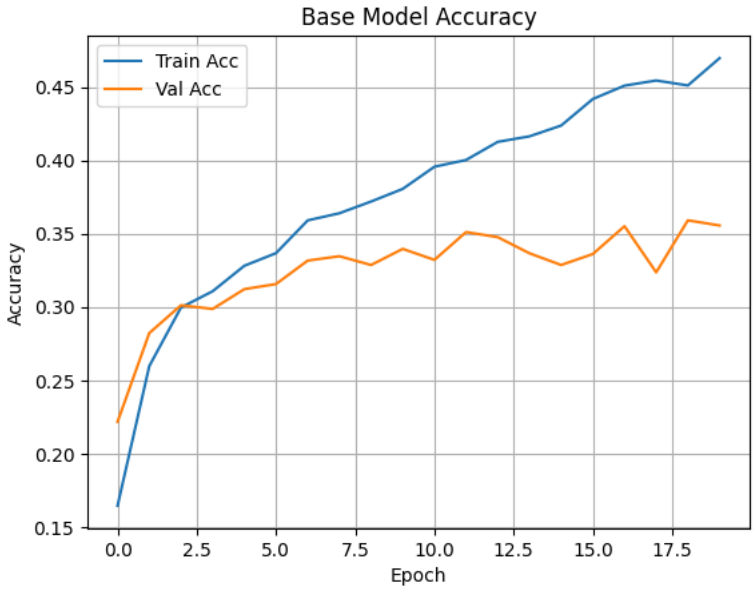

Figure 6. Base CNN model Learning Curve per epochs (20 epochs)

CNN with Dropout and Augmentation Curve

- **Observation**: The gap between the training and validation curves is smaller, indicating that overfitting is merely reduced. However, the final validation accuracy may still be lower than the baseline's final validation accuracy.
- **Interpretation**: Regularization and augmentation are preventing the model from overfitting, but possibly also slowing down learning or underfitting the data at the chosen hyperparameters.

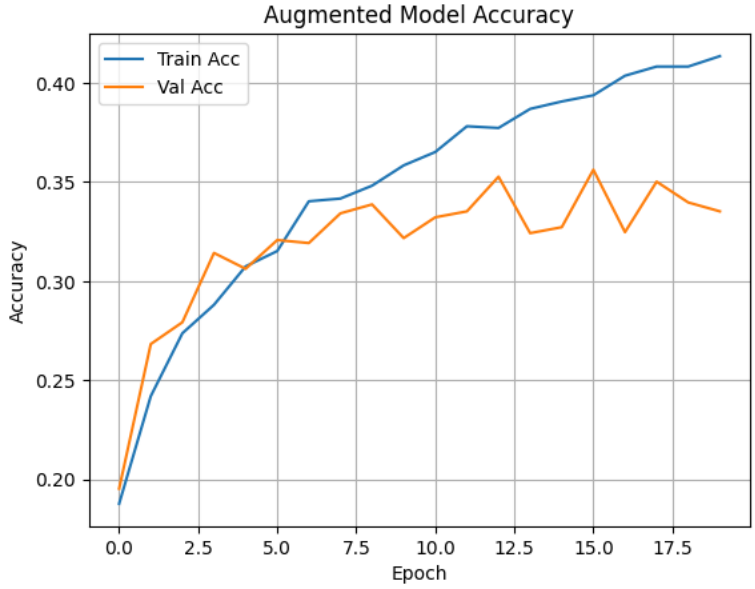

Figure 7. Augmented with Dropout CNN model Learning Curve per epochs (20 epochs)

ResNet50 Transfer Model Curve

- **Observation**: Both training and validation curves rise quickly and converge with high accuracy (~75–77%) and seeing rapid train accuracy increase from the start to epoch 2nd.
- **Interpretation:**
  - Better feature reuse
  - Stronger generalization
  - Stable learning even without early fine-tuning

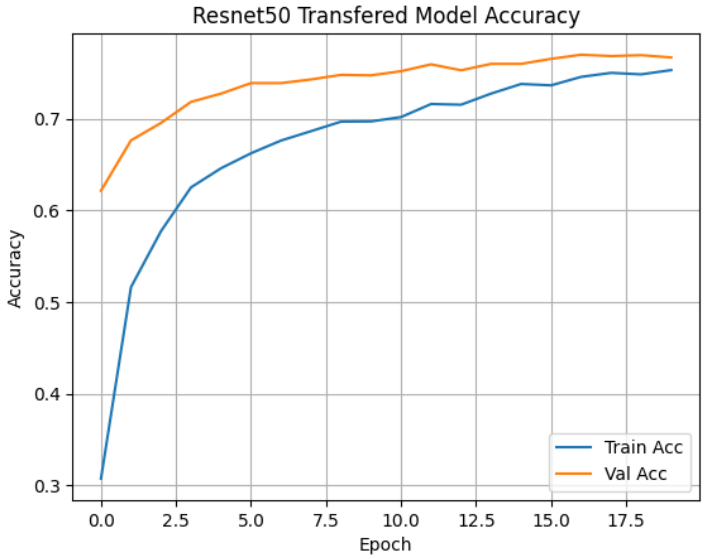

Figure 8. ResNet50 Transfer Learning model Learning Curve per epochs (20 epochs)

2.5 Overfitting Strategies

To mitigate overfitting, I implemented:

- **Data Augmentation**: Random horizontal flipping, zooming, and rotation increased training data variety.
- **Dropout Layers**: Introduced with 30% and 50% rates after convolution and dense layers respectively.
- **Smaller Model Complexity**: Avoided over-parameterizing the network.

In addition to that, ResNet50 also implicitly prevents overfitting via:

- **Frozen layers** during initial training
- **High-quality pretrained filters** learned from ImageNet
- Preprocessing normalization suited to its original training

2.6 Testing on Random Images

To test the model's generalization, I randomly selected 5 images from the test directory:

```
random_images = random.sample(os.listdir(test_path), 5)
image_paths = [os.path.join(test_path, img) for img in random_images]
```

Then, I apply image processing techniques onto it, similar to what I did when training the models, this include convert image to RGB, resize it corresponding to the model's architectures:

2.6.1. CNN bases:

Image size of (128, 128), convert the image to RGB format and NumPy array (to be passed to a model), and normalize pixel values by dividing by 255.

```
img = Image.open(image_path).convert('RGB')
img = img.resize((128, 128))
img_array = np.array(img) / 255.0
```

2.6.2. ResNet base:

Image size of (224, 224), convert the image to RGB format and NumPy array (to be passed to a model), and normalize pixel values using preprocess_resnet function.

```
img = Image.open(image_path).convert('RGB')
img = img.resize((224, 224))
img_array = np.array(img)
img_array = preprocess_resnet(img_array)
```

I have these 5 random selected testing images and the prediction results when running the 3 models:

Ground Truth: Insecta
Result: Baseline and ResNet50 are correct.

Base: Insecta | Augmented: Arachnida | ResNet50: Insecta

Ground Truth: Plantae
Result: Only ResNet50 model is correct.

Base: Aves | Augmented: Mammalia | ResNet50: Plantae

Base: Mollusca | Augmented: Amphibia | ResNet50: Amphibia

Ground Truth: Amphibia
Result: Augmented and ResNet50 are correct.

Base: Mammalia | Augmented: Mammalia | ResNet50: Mollusca

Ground Truth: Mollusca
Result: Only ResNet50 model is correct.

Base: Plantae | Augmented: Plantae | ResNet50: Plantae

Ground Truth: Plantae
Result: All models are correct.

⇒ The 5 testing images emphasize the efficiency of ResNet50 model compared to the 2 CNNs.

Conclusion

I developed 2 CNN-based models and 1 ResNet50 model with transfer learning, for a 10-class image classification task using Kaggle dataset **"Deep Learning Practice - Image Classification"**. The models were trained and evaluated with two metrics: Top-1 Accuracy and Average Accuracy per Class.

The results showed that **ResNet50 with Transfer Learning** achieved the best performance:

- Top-1 Accuracy: 76.67%
- Average Accuracy per Class: 76.67%

This demonstrates the effectiveness of transfer learning in improving model generalization, particularly when working with limited training data. Future improvements may include fine-tuning top ResNet layers, integrating other pretrained models (e.g., MobileNet, EfficientNet), and extending to multi-label tasks.

Between the 2 CNN-bases, despite regularization and augmentation, the **Baseline** model achieved slightly better results:

- Top-1 Accuracy: 35.56%
- Average Class Accuracy: 35.56%

This suggests room for tuning and further optimization, such as longer training or architectural enhancements.

Appendix

- **Tools Used**: TensorFlow/Keras, Scikit-learn, Pandas, Matplotlib
- **Evaluation Metrics**: Top-1 and Average Accuracy, Per-Class Accuracy, Classification Report
- **Data Split**: 80% Training, 20% Validation
- **Hardware**: Google Colab GPU runtime, Google Drive storage

Accessible URL from shared Google Drive, models saving directory:
https://drive.google.com/drive/folders/11Wtq-SJWMlE2vFFNlrfhip4Qt5b8CfvR?usp=sharing