**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 for a 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, we 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 CNN models, several preprocessing steps were applied using TensorFlow’s ImageDataGenerator. These include resizing, scaling, augmentation, and batching, ensuring consistency and variability during training.

**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)

**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)

**Image Resizing**

* All images were resized to a uniform dimension of 128×128 pixels using the target\_size parameter.
* This ensures consistent input dimensions across the dataset.

**1.2 Model Architectures**

We 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**

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'])

**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)

**2. Results and Discussion**

**2.1 Metrics**

We 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.

yaug\_pred\_probs = model\_augmentation.predict(val\_data)

yaug\_pred = np.argmax(yaug\_pred\_probs, axis=1)

**2.2 Evaluation Results**

|  |  |  |
| --- | --- | --- |
| **Model** | **Top-1 Accuracy** | **Avg Accuracy per Class** |
| **Baseline CNN** | 0.3556 | 0.3556 |
| **CNN + Dropout + Augment** | 0.3352 | 0.3350 |

While we 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.

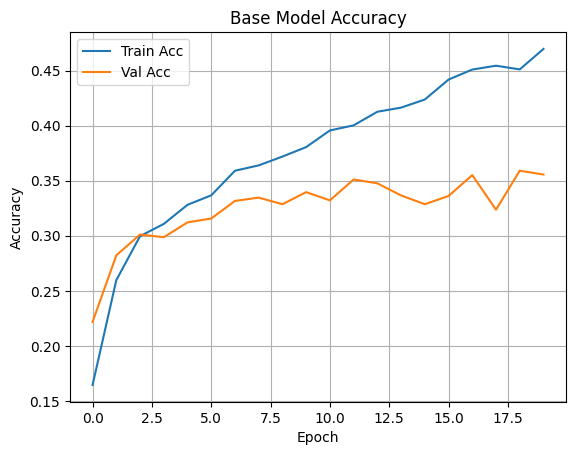
This indicates that:

* 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.

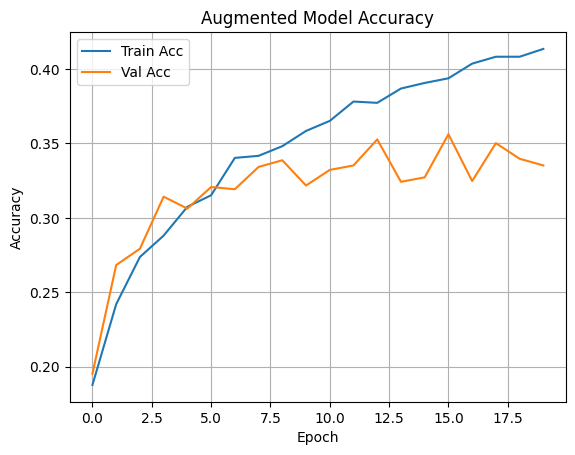
**2.3 Training Curves**

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.

**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.

**2.4 Overfitting Strategies**

We tackled overfitting by:

To mitigate overfitting, we 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.

**2.5 Testing on Random Images**

To test the model’s generalization, we 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, we apply image processing techniques onto it, similar to what we did when training the models, this include convert image to RGB, resize it (128, 128), convert the image to NumPy array (passed to a model), and normalize pixel values by dividing by 255.

img = Image.open(image\_path).convert('RGB')

img = img.resize(IMG\_SIZE)

img\_array = np.array(img) / 255.0

A close-up of a sea creature

Description automatically generatedA close up of a spider

Description automatically generatedWe have these testing images as the result when running **Baseline** and **Augmented** models

**Ground Truth**: Reptilia

**Result**: Only Augmented model is correct.

**Ground Truth**: Plantae

A hand holding a lizard

Description automatically generated**Result**: Both models are correct.

**Ground Truth**: Aves

**Result**: Both models are correct.

**Ground Truth**: Amphibia

**Result**: Both models are correct.



**Ground Truth**: Reptilia

**Result**: Only Base models is correct.

**Conclusion**

We developed two CNN-based models 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.

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 Used**: Accuracy, Per-Class Accuracy, Classification Report
* **Data Split**: 80% Training, 20% Validation
* **Hardware**: Google Colab GPU runtime