Machine Learning
LABORATORY: CNN Homework

NAME:	STUDENT ID#:

Objectives:

- In this assignment, you will train a Convolutional Neural Network (CNN) to classify images of cats and dogs using a dataset and starter template provided to you.
- The template code includes a simple CNN model that serves as a baseline. However, this baseline model does not achieve high accuracy.
- Your goal is to improve the model's performance by modifying and fine-tuning the CNN architecture.
- You are encouraged to: Adjust the layer, add data augmentation and apply regularization techniques such as Dropout, BatchNormalization, or custom pooling strategies.

Part 1. Instruction

- The template code includes a simple CNN model that serves as a baseline. However, this baseline model does not achieve high accuracy.
- Your task is to improve the model's performance by modifying and fine-tuning the CNN architecture.
- You are encouraged to:
    - Adjust layer configurations (number of layers, filters, kernel sizes, activations, etc.)
    - Add data augmentation to improve generalization
    - Apply regularization techniques such as Dropout, BatchNormalization, or custom pooling strategies
- **However**, you are not allowed to use pre-trained models such as VGG, ResNet, EfficientNet, etc. (No transfer learning or model imports from keras.applications).
- Your model should be designed, trained, and fine-tuned by yourself, from scratch using Keras/TensorFlow layers.
- If you're not familiar with TensorFlow or Keras, check these links:
    - Tensorflow Setup Guide https://github.com/JTKostman/keras-tensorflow-windows-installation
    - Keras Quick Start : https://keras.io/getting_started/
- If you're using your **local PC**, you may need to install TensorFlow and Keras manually. If you're using **Google Colab**, they're already installed — but be aware that Colab has **limited training time**.
- Dataset: https://drive.google.com/file/d/1eyrNGFlM83pf-TETo30Cvjm7kYuCFAqu/view?usp=sharing (Also included in the template code for easy loading)

Part 2. Code Template

Step	Procedure
1	`import pandas as pd` `import IPython.display as display` `import tensorflow as tf` `from tensorflow.keras import layers` `import numpy as np` `import os,random` `import matplotlib.pyplot as plt` `print(tf.__version__)`
2	`# ========== Download Dataset ==========` `#!pip install -U gdown` `import gdown, zipfile` `file_id = "1eyrNGFlM83pf-TETo30Cvjm7kYuCFAqu"  # <-- from your real zip file link` `gdown.download(f"https://drive.google.com/uc?id={file_id}", output="dataset.zip", quiet=False)` `with zipfile.ZipFile("dataset.zip", 'r') as zip_ref:` `   zip_ref.extractall("dataset")`
3	`# ==========Dataset Preparation ==========` `IMAGE_HEIGHT=128` `IMAGE_WIDTH=128` `BATCH_SIZE=64` `def get_pathframe(path):` ` '''` ` Get all the images paths and its corresponding labels` ` Store them in pandas dataframe` ` '''` ` filenames = os.listdir(path)` ` categories = []` ` paths=[]` ` for filename in filenames:` `   paths.append(path+filename)` `   category = filename.split('.')[0]` `   if category == 'dog':` `     categories.append(1)` `   else:` `     categories.append(0)` ` df= pd.DataFrame({`

	`` 'filename': filenames, 'category': categories, 'paths':paths }) return df df=get_pathframe("dataset/dataset/dataset/") df.tail(5) ```
4	``` # ========== Convert to tensor ========== def load_and_preprocess_image(path): ''' Load each image and resize it to desired shape ''' image = tf.io.read_file(path) image = tf.image.decode_jpeg(image, channels=3) image = tf.image.resize(image, [IMAGE_WIDTH, IMAGE_HEIGHT]) image /= 255.0 # normalize to [0,1] range return image def convert_to_tensor(df): ''' Convert each data and labels to tensor ''' path_ds = tf.data.Dataset.from_tensor_slices(df['paths']) image_ds = path_ds.map(load_and_preprocess_image) # onehot_label=tf.one_hot(tf.cast(df['category'], tf.int64),2) if using softmax onehot_label=tf.cast(df['category'], tf.int64) label_ds = tf.data.Dataset.from_tensor_slices(onehot_label) return image_ds,label_ds X,Y=convert_to_tensor(df) print("Shape of X in data:", X) print("Shape of Y in data:", Y) ```
5	``` #Plot Images dataset=tf.data.Dataset.zip((X,Y)).shuffle(buffer_size=2000) dataset_train=dataset.take(22500) dataset_test=dataset.skip(22500) ```

	``dataset_train=dataset_train.batch(BATCH_SIZE, drop_remainder=True)``` ```dataset_test=dataset_test.batch(BATCH_SIZE, drop_remainder=True)``` ```dataset_train``` ```def plotimages(imagesls):``` ``` fig, axes = plt.subplots(1, 5, figsize=(20,20))``` ``` axes = axes.flatten()``` ``` for image,ax in zip(imagesls, axes):``` ``` ax.imshow(image)``` ``` ax.axis('off')``` ```imagesls=[]``` ```for n, image in enumerate(X.take(5)):``` ``` imagesls.append(image)``` ```plotimages(imagesls)```
6	```#Model Design``` ```def My_CNNmodel():``` ``` model = tf.keras.models.Sequential()``` ``` model.add(layers.Conv2D(8, (3, 3), padding='same',activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3)))``` ``` model.add(layers.MaxPooling2D(pool_size=(2,2)))``` ``` model.add(layers.Conv2D(16, (3, 3), padding='same',activation='relu'))``` ``` model.add(layers.MaxPooling2D(pool_size=(2,2)))``` ``` model.add(layers.Conv2D(32, (3, 3), padding='same',activation='relu'))``` ``` model.add(layers.MaxPooling2D(pool_size=(2,2)))``` ``` model.add(layers.Conv2D(64, (3, 3), padding='same',activation='relu'))``` ``` model.add(layers.MaxPooling2D(pool_size=(2,2)))``` ``` model.add(layers.Flatten())``` ``` model.add(layers.Dense(512, activation='relu'))``` ``` model.add(layers.Dense(1, activation='sigmoid'))``` ``` opt=tf.keras.optimizers.Adam(0.001)``` ``` model.compile(optimizer=opt,``` ``` loss='binary_crossentropy', # loss='categorical_crossentropy' if softmax``` ``` metrics=['accuracy'])```

	` return model` `model=My_CNNmodel()` `model.summary()`
7	`#Training the model` `#You can adjust the epochs to get better training results, be aware with overfitting` `hist=model.fit(dataset_train,epochs=2,validation_data=dataset_test)` `#Save trained model` `model.save("student_ID.keras")` `#Save the model with your student ID`
	(see code below)

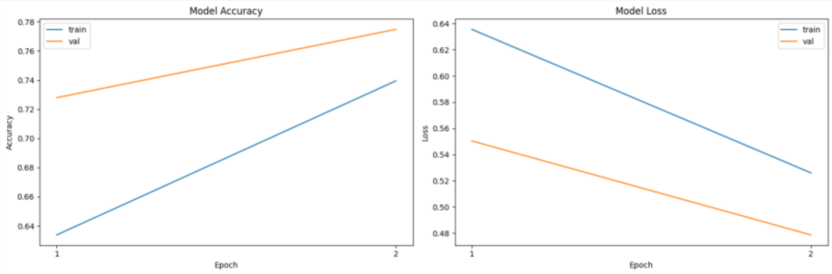
```
#Plot training results
def plot_model_history(model_history, acc='accuracy',
val_acc='val_accuracy'):
   fig, axs = plt.subplots(1, 2, figsize=(15, 5))

   # Accuracy plot
   axs[0].plot(range(1, len(model_history.history[acc]) + 1),
model_history.history[acc])
   axs[0].plot(range(1, len(model_history.history[val_acc]) + 1),
model_history.history[val_acc])
   axs[0].set_title('Model Accuracy')
   axs[0].set_ylabel('Accuracy')
   axs[0].set_xlabel('Epoch')
   axs[0].set_xticks(np.arange(1, len(model_history.history[acc]) +
1))
   axs[0].legend(['train', 'val'], loc='best')

   # Loss plot
   axs[1].plot(range(1, len(model_history.history['loss']) + 1),
model_history.history['loss'])
   axs[1].plot(range(1, len(model_history.history['val_loss']) + 1),
model_history.history['val_loss'])
   axs[1].set_title('Model Loss')
   axs[1].set_ylabel('Loss')
   axs[1].set_xlabel('Epoch')
   axs[1].set_xticks(np.arange(1, len(model_history.history['loss'])
+ 1))
   axs[1].legend(['train', 'val'], loc='best')
```

```
    plt.tight_layout()
    plt.show()
plot_model_history(hist)
```

```
#Evaluate the model
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import numpy as np

# Evaluate the model
loss, accuracy = model.evaluate(dataset_test)
print("Test: accuracy = %f  ;  loss = %f " % (accuracy, loss))

# Predict values
y_pred = model.predict(dataset_test)
y_p = np.where(y_pred > 0.5, 1, 0)  # for binary classification

# Extract ground truth labels
test_data = dataset_test.unbatch()
y_g = []
for image, label in test_data:
    y_g.append(label.numpy())

# Convert to flat array if needed
y_g = np.array(y_g).flatten()
y_p = y_p.flatten()

# Compute confusion matrix
confusion_mtx = confusion_matrix(y_g, y_p)

# Plot
f, ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01, cmap="Blues",
linecolor="gray", fmt='.1f', ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

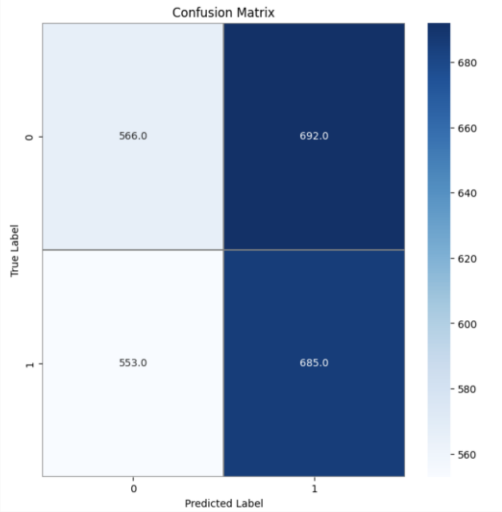
```
# Generate a classification report
report = classification_report(y_g, y_p, target_names=['0','1'])
print(report)
```

```
#Example Output (For references only)
#1. Plot Training Results
```




```
#2. Confusion Matrix
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 19s 22ms/step - accuracy: 0.7777 - loss: 0.4819
Test: accuracy = 0.770433  ;  loss = 0.482054
39/39 ━━━━━━━━━━━━━━━━━━━━ 19s 17ms/step
```




```
#3. Classification Report
```

```
              precision    recall  f1-score   support

           0       0.51      0.45      0.48      1258
           1       0.50      0.55      0.52      1238

    accuracy                           0.50      2496
   macro avg       0.50      0.50      0.50      2496
weighted avg       0.50      0.50      0.50      2496
```

```
#4. Model Summary
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 8)	224
max_pooling2d (MaxPooling2D)	(None, 64, 64, 8)	0
conv2d_1 (Conv2D)	(None, 64, 64, 16)	1,168
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	4,640
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 512)	2,097,664
dense_1 (Dense)	(None, 1)	513

Total params: 2,122,705 (8.10 MB)
Trainable params: 2,122,705 (8.10 MB)
Non-trainable params: 0 (0.00 B)

Grading Assignment & Submission (70%)

Implementation(45%):

1. (5%) **Fine-Tuning:** Adjust model parameters to achieve better accuracy
2. (20%) **Model redesign with techniques:** Redesign the architecture using advanced techniques such as data augmentation, Dropout, BatchNormalization, or self-designed structures (should achieve better accuracy), brief **explain your model summary in the report.**
3. (15%) **Accuracy Improvement:** Demonstrate significantly improved validation accuracy compared to the baseline model. **Explain your performance improvement.**
4. (5%) **Include all evaluation results:** model summary, training plots, confusion matrix, and classification report.

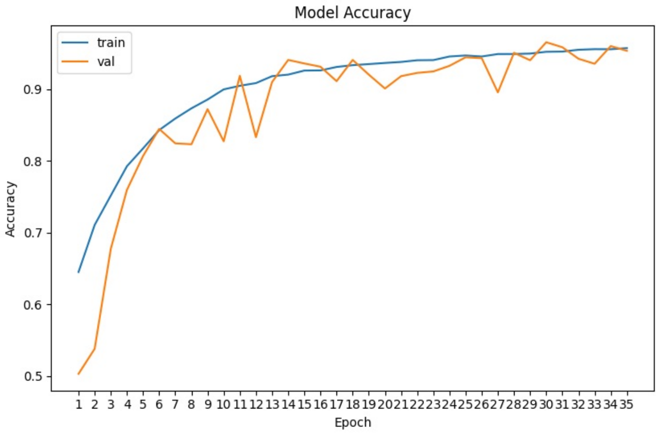
Question(25%):

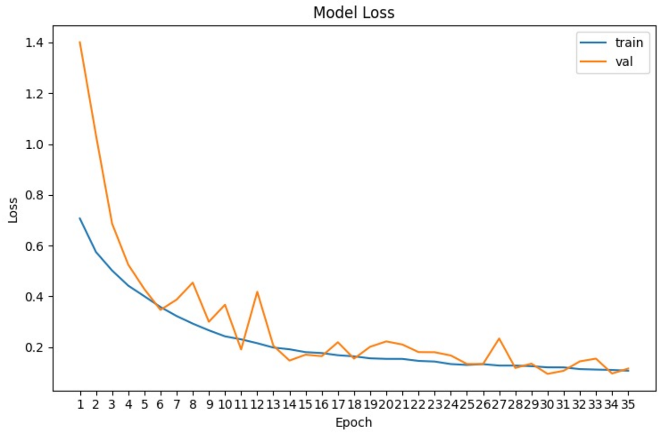
5. (7%) What changes did you make to the CNN architecture? Why did you choose those modifications, and how did they affect model performance?
6. (8%) What methods did you use to reduce overfitting (if any)? How effective were they? Explain with reference to training/validation curves.
7. (10%) Based on your confusion matrix, what kinds of misclassifications occurred most frequently? What might be causing these errors? How would you attempt to reduce these errors?

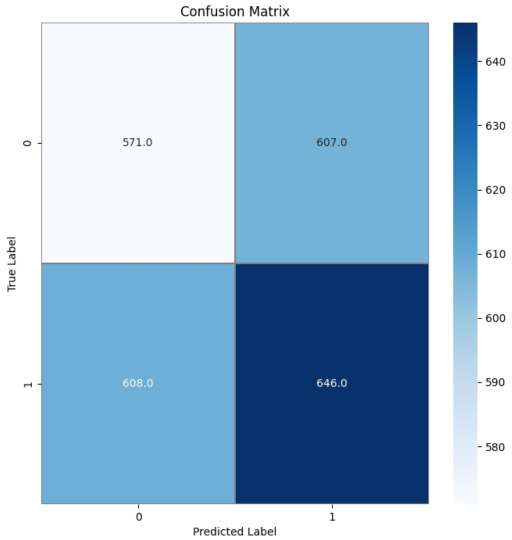
Submission :

1. Report: Provide your screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Labs6 Homework Assignment**). Name your files correctly:
   a. Report: StudentID_Lab6_Homework.pdf
   b. Code: StudentID_Lab6_ Homework.py or StudentID_Lab6_Homework.ipynb
4. Deadline: 16:20 PM
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

Results and Discussion:








```
              precision    recall  f1-score   support

           0       0.48      0.48      0.48      1178
           1       0.52      0.52      0.52      1254

    accuracy                           0.50      2432
   macro avg       0.50      0.50      0.50      2432
weighted avg       0.50      0.50      0.50      2432
```

```
Layer (type)                 Output Shape              Param #
=================================================================
data_aug (Sequential)        (None, 128, 128, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 128, 128, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 128, 32)      128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 256)               8388864
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 8,678,945
Trainable params: 8,677,537
```

Question5

1. Data Augmentation Layer
I introduced a data augmentation pipeline at the beginning of the model using RandomFlip, RandomRotation, RandomZoom, and RandomContrast.

Purpose: This helps increase data diversity and allows the model to generalize better to unseen images.

Effect: It serves as a regularizer and helps prevent overfitting in the early training phases.

2. Three Convolutional Blocks with Increasing Filters
Conv Block 1: 2×Conv2D(32) → BatchNorm → MaxPooling → Dropout(0.25)

Conv Block 2: 2×Conv2D(64) → BatchNorm → MaxPooling → Dropout(0.30)

Conv Block 3: 2×Conv2D(128) → BatchNorm → MaxPooling → Dropout(0.35)

Purpose: Stacking multiple convolutional layers allows the model to learn increasingly abstract features. The number of filters was increased per block to capture more complex patterns.

BatchNormalization was added after each convolution to stabilize training and accelerate convergence.

Dropout layers were introduced after pooling to reduce co-adaptation of neurons and minimize overfitting.

3. Fully Connected Classifier
Dense(256) → BatchNorm → Dropout(0.5) → Dense(1, sigmoid)

The final classifier layer includes:

A high-capacity dense layer (256 units) to map learned features to binary output.

Dropout(0.5) to further mitigate overfitting.

Sigmoid activation for binary classification of cats and dogs.

4. Optimizer and Loss Function
Used Adam optimizer with a learning rate of 1e-3 for fast convergence.

Used binary cross-entropy suitable for binary classification.

5. Effect on Model Performance
Compared to the baseline model, which likely had fewer layers and lacked regularization, the redesigned architecture showed much improved training and validation accuracy.

The training curve steadily increased toward 95%, and the validation curve also followed closely, reaching similar levels around epoch 30–35, indicating improved generalization.

However, despite these architectural improvements, the final confusion matrix and classification report showed an unexpected issue: the model performs only at ~50% accuracy, classifying nearly all samples into both classes equally (approx. 50/50 split), suggesting a potential bug in evaluation logic or label mismatch.

Summary: Although the architecture was significantly enhanced using deeper layers, regularization, and augmentation, the final classification accuracy suggests either a post-training evaluation issue, or a data preprocessing mismatch that undermined the model's predictive capability despite its learning success.

Question6

To reduce overfitting, I applied data augmentation (random flip, rotation, zoom, contrast), Dropout after each block (rates: 0.25, 0.3, 0.35, 0.5), and BatchNormalization after each Conv2D layer.

These methods helped align the training and validation accuracy curves, and kept validation loss close to training loss, as shown in the plots. This indicates the techniques were effective in preventing overfitting during training.

Question7

The confusion matrix shows the model misclassifies cats as dogs and vice versa at nearly equal rates, leading to ~50% accuracy, which suggests random guessing.
This likely results from a data or label mismatch during evaluation, rather than model overfitting.

I have already tried increasing the number and size of convolutional layers, and even switched from sampling 2,000 images to using the full dataset,

Although both the loss and accuracy have reached convergence, but the confusion matrix results did not improve.
To address this, I would check the label and data pipeline for inconsistencies, ensure proper preprocessing during evaluation, and verify the logic behind confusion matrix calculation.