1. import tensorflow as tf from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Dense model = Sequential() model.add(Dense(4,
activation='relu', input_shape=(8,))) model.add(Dense(8,
activation='relu')) model.add(Dense(4, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(
optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy']
)
model.summary()
from tensorflow.keras.utils import plot_model plot_model(model,
to_file='Problem_1.png', show_shapes=True) print("\nModel
successfully created.")

2.
import tensorflow as tf from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Dense model = Sequential() model.add(Dense(12,
activation='relu', input_shape=(6,))) model.add(Dense(8,
activation='relu')) model.add(Dense(3,
activation='softmax')) model.compile(
optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']
)
model.summary()

```
from tensorflow.keras.utils import plot_model plot_model(model,
to_file='Problem_2.png', show_shapes=True)
print("Model successfully built.")
```

3.

```
import numpy as np import matplotlib.pyplot as plt
import tensorflow as tf from
sklearn.model_selection import train_test_split x =
np.linspace(-10, 10, 1000) x = x.reshape(-1,1) def
linear(x):
   return 5*x + 10 def
quadratic(x):
   return 3*x**2 + 5*x + 10 def
cubic(x):    return 4*x**3 + 3*x**2 +
5*x + 10 y = quadratic(x)
X_train, X_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42) X_val,
X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
model = tf.keras.Sequential([ tf.keras.layers.Dense(64, activation='relu', input_shape=(1,)),
tf.keras.layers.Dense(64, activation='relu'), tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse') history
= model.fit(X_train, y_train,
                validation_data=(X_val, y_val),
epochs=200,                 batch_size=32,
verbose=1)
y_pred = model.predict(X_test) plt.scatter(X_test,
y_test, label="Original") plt.scatter(X_test,
y_pred, label="Predicted") plt.legend()
```

```
plt.title("Original vs Predicted")
plt.savefig("Problem_3_quadratic.png")
plt.show() print("Test Loss:",
model.evaluate(X_test, y_test))
```

4.

```
import tensorflow as tf from tensorflow.keras.models import
Sequential from tensorflow.keras.layers import Dense, Flatten,
Dropout from tensorflow.keras.datasets import fashion_mnist,
mnist, cifar10 import matplotlib.pyplot as plt import os
os.makedirs("plots", exist_ok=True) def build_model(input_shape):
    model = Sequential([
        Flatten(input_shape=input_shape),
        Dense(256, activation='relu'),
        Dropout(0.3),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(        optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy']
    )
    return model def
train_dataset(dataset_name, dataset):
    (X_train, y_train), (X_test, y_test) = dataset
    X_train = X_train / 255.0     X_test =
X_test / 255.0     model =
```

```
build_model(X_train.shape[1:])     history
= model.fit(        X_train, y_train,
epochs=20,        batch_size=64,
validation_split=0.2,        verbose=2
   )
   loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"{dataset_name} Test Accuracy: {acc*100:.2f}%")
plt.figure(figsize=(8,6)) plt.plot(history.history['accuracy'],
label='Train Accuracy') plt.plot(history.history['val_accuracy'],
label='Validation Accuracy') plt.title(f'{dataset_name} Accuracy')
   plt.xlabel('Epochs') plt.ylabel('Accuracy') plt.legend()
plt.grid(True) plot_path = f"plots/{dataset_name.replace(' ',
'_')}_accuracy.png"     plt.savefig(plot_path)     plt.show()
print(f"Plot saved to {plot_path}\n") train_dataset("Fashion
MNIST", fashion_mnist.load_data()) train_dataset("MNIST",
mnist.load_data()) train_dataset("CIFAR-10", cifar10.load_data())
```

5.

```
import tensorflow as tf from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout from
tensorflow.keras.datasets import fashion_mnist, mnist, cifar10 import
matplotlib.pyplot as plt import os os.makedirs("plots", exist_ok=True) def
build_cnn(input_shape):
   model = Sequential([
      Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
      MaxPooling2D((2,2)),
      Conv2D(64, (3,3), activation='relu'),
      MaxPooling2D((2,2)),
```

```
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
    return model def train_and_plot(dataset_name,
dataset, is_cifar=False):
    (X_train, y_train), (X_test, y_test) = dataset
    X_train = X_train / 255.0
X_test = X_test / 255.0
    if not is_cifar:
        X_train = X_train[..., tf.newaxis]
X_test = X_test[..., tf.newaxis]
    model = build_cnn(X_train.shape[1:])
history = model.fit(        X_train,
y_train,        epochs=20,
batch_size=64,
validation_split=0.2,        verbose=2
    )
    loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"{dataset_name} Test Accuracy: {acc*100:.2f}%")
plt.figure(figsize=(8,6))   plt.plot(history.history['accuracy'],
label='Train Accuracy') plt.plot(history.history['val_accuracy'],
label='Validation Accuracy') plt.title(f'{dataset_name} Accuracy')
plt.xlabel('Epochs') plt.ylabel('Accuracy')
```

```
    plt.legend() plt.grid(True) plot_path =
f"plots/{dataset_name.replace(' ', '_')}_Problem_5.png"
plt.savefig(plot_path) plt.show()     print(f"Plot saved to
{plot_path}\n") train_and_plot("Fashion MNIST",
fashion_mnist.load_data()) train_and_plot("MNIST",
mnist.load_data()) train_and_plot("CIFAR-10", cifar10.load_data(),
is_cifar=True)
```

6.

```
import numpy as np import tensorflow as tf import
matplotlib.pyplot as plt from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Dense, Flatten, Dropout from tensorflow.keras.datasets
import mnist from sklearn.model_selection import
train_test_split
(X_train_mnist, y_train_mnist), (X_test_mnist, y_test_mnist) = mnist.load_data()
X_train_mnist = X_train_mnist / 255.0
X_test_mnist = X_test_mnist / 255.0
X_custom = X_train_mnist[:5000] y_custom
= y_train_mnist[:5000]
X_custom_train, X_custom_test, y_custom_train, y_custom_test = train_test_split(
    X_custom, y_custom, test_size=0.2, random_state=42
)
print("Custom Training Samples:", X_custom_train.shape[0]) print("Custom
Testing Samples:", X_custom_test.shape[0]) X_combined_train =
np.concatenate((X_train_mnist, X_custom_train), axis=0)
y_combined_train = np.concatenate((y_train_mnist, y_custom_train),
```

```
axis=0) print("Total Combined Training Samples:",
X_combined_train.shape[0]) model = Sequential([
   Flatten(input_shape=(28, 28)),
   Dense(256, activation='relu'),
   Dropout(0.3),
   Dense(128, activation='relu'),
   Dropout(0.3),
   Dense(64, activation='relu'),
   Dense(10, activation='softmax')
])
model.compile(
   optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy']
)
model.summary()
history = model.fit(
X_combined_train,
y_combined_train,
epochs=20,    batch_size=64,
validation_split=0.2
)
custom_loss, custom_acc = model.evaluate(X_custom_test, y_custom_test)
print("\nAccuracy on Custom Test Dataset: {:.2f}%".format(custom_acc * 100))
mnist_loss, mnist_acc = model.evaluate(X_test_mnist, y_test_mnist) print("Accuracy
on MNIST Test Dataset: {:.2f}%".format(mnist_acc * 100)) plt.figure()
plt.plot(history.history['accuracy']) plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy') plt.ylabel('Accuracy') plt.xlabel('Epoch') plt.legend(['Train',
'Validation']) plt.savefig("accuracy_plot.png", dpi=300) plt.show() plt.figure()
```

```
plt.plot(history.history['loss']) plt.plot(history.history['val_loss']) plt.title('Model
Loss') plt.ylabel('Loss') plt.xlabel('Epoch') plt.legend(['Train', 'Validation'])
plt.savefig("loss_plot.png", dpi=300) plt.show() custom_accuracy = custom_acc *
100 mnist_accuracy = mnist_acc * 100
plt.figure() plt.bar(['Custom Dataset', 'MNIST Dataset'],
[custom_accuracy, mnist_accuracy]) plt.ylabel('Accuracy (%)')
plt.title('Custom vs MNIST Test Accuracy') plt.text(0,
custom_accuracy + 0.5, f"{custom_accuracy:.2f}%") plt.text(1,
mnist_accuracy + 0.5, f"{mnist_accuracy:.2f}%")
plt.savefig("dataset_accuracy_comparison.png", dpi=300)
plt.show()
print("\nFigures Saved:") print("1.
accuracy_plot.png") print("2. loss_plot.png")
print("3. dataset_accuracy_comparison.png")
```

7.

```
import tensorflow as tf import numpy as np import matplotlib.pyplot as plt import
time import os from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout from
tensorflow.keras.preprocessing.image import ImageDataGenerator
IMG_SIZE = 128
BATCH_SIZE = 32
EPOCHS = 15
DATASET_DIR = 'datasets/'  # Folder containing 'resizedShirt' and 'Ishirt' subfolders
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2) train_data =
datagen.flow_from_directory(    DATASET_DIR,    target_size=(IMG_SIZE,
IMG_SIZE),    batch_size=BATCH_SIZE,    subset='training',    class_mode='binary'
# Binary classification
)
```

```
test_data = datagen.flow_from_directory(
DATASET_DIR,       target_size=(IMG_SIZE,
IMG_SIZE),                    batch_size=1,
subset='validation',
class_mode='binary',     shuffle=False
)
num_classes = 2  # Only two classes def
build_model(filters1=32, filters2=64, filters3=128):
   model = Sequential([
      Conv2D(filters1, (3,3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
MaxPooling2D(2,2),

      Conv2D(filters2, (3,3), activation='relu'),
      MaxPooling2D(2,2),

      Conv2D(filters3, (3,3), activation='relu'),
      MaxPooling2D(2,2),

      Flatten(),
      Dense(128, activation='relu'),
      Dropout(0.3),
      Dense(1, activation='sigmoid')  # Binary classification output
   ])

   model.compile(optimizer='adam',
              loss='binary_crossentropy',
metrics=['accuracy'])
   return model model
=           build_model()
```

```
model.summary()
start_train = time.time() history = model.fit(train_data, epochs=EPOCHS)
end_train = time.time()
training_time = end_train - start_train
print("Total Training Time (seconds):", training_time)
start_test = time.time() loss, accuracy = model.evaluate(test_data) end_test = time.time()
testing_time = end_test - start_test
time_per_sample = testing_time / test_data.samples
print("Test Accuracy:", accuracy) print("Total Testing Time (seconds):", testing_time)
```

```
plt.figure() plt.plot(history.history['accuracy'],
label='Train Accuracy') plt.plot(history.history['loss'],
label='Train Loss') plt.title("Epoch vs Accuracy / Loss")
plt.xlabel("Epoch") plt.ylabel("Value") plt.legend()
plt.grid(True)
plt.savefig("epoch_vs_accuracy_binary.png", dpi=300)
plt.show() sizes = [0.25, 0.5, 0.75, 1.0] accuracies = [] for s
in sizes:
   train_subset = datagen.flow_from_directory(
DATASET_DIR,        target_size=(IMG_SIZE,
IMG_SIZE),        batch_size=BATCH_SIZE,
      subset='training',
class_mode='binary'
   )
   model_temp = build_model()
model_temp.fit(train_subset, epochs=5, verbose=0)     _,
acc_temp = model_temp.evaluate(test_data, verbose=0)
accuracies.append(acc_temp)
plt.figure() plt.plot(sizes, accuracies)
plt.xlabel("Fraction of Training Data")
plt.ylabel("Accuracy") plt.title("Data Size vs
Accuracy") plt.grid(True)
plt.savefig("data_vs_accuracy_binary.png", dpi=300)
plt.show() filter_configs = [(16,32,64), (32,64,128),
(64,128,256)] param_counts = [] model_accs = [] for f
in filter_configs:     m = build_model(*f)
param_counts.append(m.count_params())
```

```
   m.fit(train_data,   epochs=5,   verbose=0)
_,  acc  =  m.evaluate(test_data,  verbose=0)
model_accs.append(acc)
plt.figure() plt.plot(param_counts,
model_accs) plt.xlabel("Number of
Parameters") plt.ylabel("Accuracy")
plt.title("Model Size vs Accuracy")
plt.grid(True)
plt.savefig("modelsize_vs_accuracy
_binary.png", dpi=300) plt.show()
```

8.

```
import tensorflow as tf import matplotlib.pyplot as plt from
tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Conv2D, MaxPooling2D from
tensorflow.keras.layers import Flatten, Dense, Dropout from
tensorflow.keras.datasets import cifar10 from
tensorflow.keras.utils import to_categorical
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train / 255.0 X_test = X_test / 255.0 y_train =
to_categorical(y_train, 10) y_test = to_categorical(y_test, 10)
model = Sequential() model.add(Conv2D(64, (3,3),
activation='relu', padding='same',
            input_shape=(32,32,3)))
model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2))) model.add(Conv2D(128, (3,3),
activation='relu', padding='same')) model.add(Conv2D(128, (3,3),
activation='relu', padding='same'))
```

```
model.add(MaxPooling2D((2,2))) model.add(Conv2D(256, (3,3),
activation='relu', padding='same')) model.add(Conv2D(256, (3,3),
activation='relu', padding='same')) model.add(Conv2D(256, (3,3),
activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2))) model.add(Conv2D(512, (3,3),
activation='relu', padding='same')) model.add(Conv2D(512, (3,3),
activation='relu', padding='same')) model.add(Conv2D(512, (3,3),
activation='relu', padding='same'))
model.add(MaxPooling2D((2,2))) model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5)) model.add(Dense(4096,
activation='relu')) model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax')) model.compile(
   optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy']
)
model.summary()
history = model.fit(
X_train, y_train,
epochs=10,
batch_size=64,
validation_split=0.2
)
```

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print("\nFinal Test Accuracy:", test_acc) final_train_acc
= history.history['accuracy'][-1] final_val_acc =
history.history['val_accuracy'][-1] print("Final Training
Accuracy:", final_train_acc) print("Final Validation
Accuracy:", final_val_acc) plt.figure()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epoch') plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
plt.savefig("vgg_epoch_accuracy.png", dpi=300) plt.show()
plt.figure() plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch') plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.savefig("vgg_epoch_loss.png", dpi=300)
plt.show()
```

9.

```
import numpy as np import matplotlib.pyplot as plt import tensorflow as tf
from tensorflow.keras.preprocessing import image from
tensorflow.keras.applications import VGG16, ResNet50, MobileNetV2 from
tensorflow.keras.applications.vgg16 import preprocess_input as vgg_pre from
tensorflow.keras.applications.resnet50 import preprocess_input as res_pre
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input as
mob_pre from tensorflow.keras.models import Model import os
os.makedirs("plots", exist_ok=True) img_path = "image.jpg"  # Replace with
```

```
your image path img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img) img_array =
np.expand_dims(img_array, axis=0)  # Shape: (1, H, W, C) def
visualize_feature_maps(model, preprocess_fn, save_name,
max_channels=16):     img_processed = preprocess_fn(img_array.copy())
conv_layers = [layer.output for layer in model.layers if "conv" in layer.name]
if len(conv_layers) == 0:
        print(f"No convolution layers found in {model.name}")
        return
    layer_output = conv_layers[0]  # first conv layer only
feature_model = Model(inputs=model.input, outputs=layer_output)
feature_maps = feature_model.predict(img_processed)     fmap =
feature_maps[0]  # shape: (H, W, C)     num_channels = fmap.shape[-
1]     num_to_plot = min(max_channels, num_channels)
    plt.figure(figsize=(10, 10))
for i in range(num_to_plot):
        plt.subplot(4, 4, i+1)
plt.imshow(fmap[:, :, i], cmap='viridis')
plt.axis('off')


    plt.tight_layout()
    plt.savefig(f"plots/{save_name}", dpi=300)
    plt.show()     print(f"Feature map plot saved as:
    plots/{save_name}") vgg_model =
    VGG16(weights='imagenet', include_top=False)
    visualize_feature_maps(vgg_model, vgg_pre,
    "vgg_feature_maps.png") resnet_model =
    ResNet50(weights='imagenet', include_top=False)
```

```
    visualize_feature_maps(resnet_model, res_pre,
    "resnet_feature_maps.png") mobilenet_model =
    MobileNetV2(weights='imagenet',
    include_top=False)
    visualize_feature_maps(mobilenet_model,
    mob_pre, "mobilenet_feature_maps.png")
```

10.
```
import tensorflow as tf from tensorflow.keras.preprocessing.image
import ImageDataGenerator from tensorflow.keras.models import
Model from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.applications import VGG16 import
matplotlib.pyplot as plt import os
IMG_SIZE = 128
BATCH_SIZE = 32
EPOCHS = 15
DATASET_DIR = 'datasets/'  # Folder containing 'resizedShirt' and 'Ishirt'
os.makedirs("plots", exist_ok=True) datagen =
ImageDataGenerator(rescale=1./255, validation_split=0.2) train_data =
datagen.flow_from_directory(
   DATASET_DIR,
 target_size=(IMG_SIZE, IMG_SIZE),
 batch_size=BATCH_SIZE,
 subset='training', class_mode='binary'
)
val_data = datagen.flow_from_directory(
DATASET_DIR,     target_size=(IMG_SIZE,
IMG_SIZE),     batch_size=BATCH_SIZE,
```

```
    subset='validation',
class_mode='binary'
)
def build_vgg16_model(trainable_layers='all'):
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
    if trainable_layers == 'top':         for layer in base_model.layers[:-4*3]:  #
Approx 4 conv blocks = 4*3 layers
            layer.trainable = False
else:
        # Whole model trainable
for layer in base_model.layers:
layer.trainable = True
    x = Flatten()(base_model.output)     x = Dense(128, activation='relu')(x)     x =
Dropout(0.3)(x)     output = Dense(1, activation='sigmoid')(x)     model =
Model(inputs=base_model.input, outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
return model def train_and_plot(trainable_type):    if trainable_type == 'all':
       model_name = "vgg16_whole_finetune"
   print("\n=== Training: Whole VGG16 Fine-tuning ===")
   else:
        model_name = "vgg16_partial_finetune"         print("\n===
Training: Partial VGG16 Fine-tuning ===")     model =
build_vgg16_model(trainable_layers=trainable_type)
    history = model.fit(
train_data,
validation_data=val_data,
epochs=EPOCHS,         verbose=2
    )
```

```
   loss, acc = model.evaluate(val_data, verbose=0)
print(f"{model_name} Test Accuracy: {acc*100:.2f}%")
plt.figure(figsize=(8,6))     plt.plot(history.history['accuracy'],
label='Train Accuracy')     plt.plot(history.history['val_accuracy'],
label='Validation Accuracy')     plt.title(f"{model_name} Accuracy")
plt.xlabel("Epochs")     plt.ylabel("Accuracy")     plt.legend()
plt.grid(True)
   plot_path = f"plots/{model_name}.png"
plt.savefig(plot_path, dpi=300)     plt.show()     print(f"Plot
saved to {plot_path}\n")     return acc acc_whole =
train_and_plot('all') acc_partial = train_and_plot('top')
print("Final Test Accuracy:") print(f"Whole VGG16 Fine-
tuning: {acc_whole*100:.2f}%") print(f"Partial VGG16
Fine-tuning: {acc_partial*100:.2f}%")
```

11.

```
import tensorflow as tf import numpy as np import
matplotlib.pyplot as plt from tensorflow.keras.datasets
import mnist from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.applications import VGG16 from
sklearn.decomposition import PCA from sklearn.manifold
import TSNE import os
IMG_SIZE = 32  # Resize MNIST digits to 32x32
BATCH_SIZE  =  128  EPOCHS  =  5
os.makedirs("plots", exist_ok=True)
(X_train, y_train), (X_test, y_test) = mnist.load_data() def
preprocess_mnist(X):
```

```
   X_resized = np.stack([tf.image.resize(tf.expand_dims(img, -1), [IMG_SIZE, IMG_SIZE]).numpy() for img in X], axis=0)
   X_rgb = np.repeat(X_resized, 3, axis=-1)  # Convert grayscale to RGB
X_rgb = X_rgb / 255.0     return X_rgb
X_train_proc = preprocess_mnist(X_train) X_test_proc = preprocess_mnist(X_test) base_model = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
base_model.trainable = False  # Feature extraction only x = Flatten()(base_model.output) x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x) output = Dense(10, activation='softmax')(x) model = Model(inputs=base_model.input, outputs=output) model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train_proc, y_train, validation_split=0.2, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=2) def extract_features(model, X):
   feature_model = Model(inputs=model.input, outputs=model.layers[-3].output)  # Flatten layer
features = feature_model.predict(X, batch_size=BATCH_SIZE, verbose=1)     num_samples = features.shape[0]     features_flat = features.reshape(num_samples, -1)  # Flatten to (num_samples, H*W*C)
   return features_flat
features_before = extract_features(Model(inputs=base_model.input, outputs=Flatten()(base_model.output)), X_test_proc) features_after = extract_features(model, X_test_proc) def plot_2d(features, labels, method='PCA', filename='plot.png'):     if method == 'PCA':
      reducer = PCA(n_components=2)
elif method == 'TSNE':
      reducer = TSNE(n_components=2, random_state=42, perplexity=30, max_iter=500)
   else:
      raise ValueError("Method must be 'PCA' or 'TSNE'")
features_2d = reducer.fit_transform(features)
```

```
   plt.figure(figsize=(8,6))
for i in range(10):
      plt.scatter(features_2d[labels==i,0], features_2d[labels==i,1], label=str(i), alpha=0.6)
plt.title(f"{method} Visualization of Features")     plt.xlabel("Component 1")
plt.ylabel("Component 2")
   plt.legend()     plt.grid(True)     plt.savefig(f"plots/{filename}", dpi=300)     plt.show()
print(f"Saved plot: plots/{filename}") plot_2d(features_before, y_test, method='PCA',
filename='features_before_PCA.png') plot_2d(features_before, y_test, method='TSNE',
filename='features_before_tSNE.png') plot_2d(features_after, y_test, method='PCA',
filename='features_after_PCA.png') plot_2d(features_after, y_test, method='TSNE',
filename='features_after_tSNE.png')
```

12.

```
import os os.environ["TF_ENABLE_ONEDNN_OPTS"] = "1" import tensorflow as tf
import numpy as np import matplotlib.pyplot as plt from
tensorflow.keras.datasets import cifar10 from tensorflow.keras.models import
Sequential from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Input from tensorflow.keras.preprocessing.image import
ImageDataGenerator from tensorflow.keras.utils import to_categorical
tf.config.threading.set_intra_op_parallelism_threads(0)
tf.config.threading.set_inter_op_parallelism_threads(0)
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train[:20000] y_train = y_train[:20000]
X_train = X_train / 255.0
X_test = X_test / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
def build_model():     model =
Sequential([
       Input(shape=(32,32,3)),
       Conv2D(16, (3,3), activation='relu'),
       MaxPooling2D(2,2),
       Conv2D(32, (3,3), activation='relu'),
       MaxPooling2D(2,2),
       Flatten(),
       Dense(64, activation='relu'),
       Dense(10, activation='softmax')
   ])
   model.compile(optimizer='adam',
                 loss='categorical_crossentropy',
metrics=['accuracy'])     return model
EPOCHS = 10 BATCH = 128
model_no_aug = build_model()
history_no_aug = model_no_aug.fit(
X_train, y_train,
   epochs=EPOCHS,
batch_size=BATCH,
   validation_split=0.2,
verbose=1
)
val_acc_no_aug = history_no_aug.history['val_accuracy'][-1] flip_gen =
ImageDataGenerator(horizontal_flip=True, validation_split=0.2)
model_flip = build_model() history_flip = model_flip.fit(
flip_gen.flow(X_train, y_train, batch_size=BATCH, subset='training'),
epochs=EPOCHS,     validation_data=flip_gen.flow(X_train, y_train,
batch_size=BATCH, subset='validation'),     verbose=1
```

```
)
val_acc_flip = history_flip.history['val_accuracy'][-1] rot_gen =
ImageDataGenerator(rotation_range=15, validation_split=0.2) model_rot =
build_model() history_rot = model_rot.fit(     rot_gen.flow(X_train, y_train,
batch_size=BATCH, subset='training'),     epochs=EPOCHS,
validation_data=rot_gen.flow(X_train, y_train, batch_size=BATCH, subset='validation'),
verbose=1
)
val_acc_rot = history_rot.history['val_accuracy'][-1]
combo_gen = ImageDataGenerator(
   horizontal_flip=True,
rotation_range=15,     zoom_range=0.1,
width_shift_range=0.1,
height_shift_range=0.1,
validation_split=0.2
)
model_combo = build_model() history_combo = model_combo.fit(
combo_gen.flow(X_train, y_train, batch_size=BATCH, subset='training'),
epochs=EPOCHS,
   validation_data=combo_gen.flow(X_train, y_train, batch_size=BATCH, subset='validation'),
verbose=1
)
val_acc_combo = history_combo.history['val_accuracy'][-1] methods =
["No Aug", "Flip", "Rotation", "Combined"] accuracies =
[val_acc_no_aug, val_acc_flip, val_acc_rot, val_acc_combo] plt.figure()
plt.bar(methods, accuracies) plt.ylabel("Validation
Accuracy") plt.title("Effect of Data Augmentation")
plt.savefig("augmentation_comparison.png", dpi=300)
```

```
plt.show() print("\nValidation Accuracies:") for m, a in
zip(methods, accuracies):
   print(f"{m}: {a:.4f}")
```

13.

```
import os os.environ["TF_ENABLE_ONEDNN_OPTS"]
= "1"
import tensorflow as tf import numpy as np import matplotlib.pyplot as plt from
tensorflow.keras.datasets import cifar10 from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator from
tensorflow.keras.utils import to_categorical
tf.config.threading.set_intra_op_parallelism_threads(0)
tf.config.threading.set_inter_op_parallelism_threads(0)

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train[:25000] y_train = y_train[:25000]
X_train = X_train / 255.0 X_test = X_test / 255.0
y_train = to_categorical(y_train, 10) y_test =
to_categorical(y_test, 10)
EPOCHS = 10 BATCH = 128 def
build_model(use_dropout=False):
   model = Sequential([
      Input(shape=(32,32,3)),
      Conv2D(16, (3,3), activation='relu'),
      MaxPooling2D(2,2),
      Conv2D(32, (3,3), activation='relu'),
      MaxPooling2D(2,2),
```

```
    Flatten(),
    Dense(64, activation='relu')
  ])
  if use_dropout:
    model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
metrics=['accuracy'])
  return model model_A =
build_model(False) history_A
= model_A.fit( X_train,
y_train,
  epochs=EPOCHS,
batch_size=BATCH,
  validation_split=0.2,
verbose=1
)
model_B = build_model(True)
history_B = model_B.fit(
X_train, y_train,
epochs=EPOCHS,
batch_size=BATCH,
validation_split=0.2,
verbose=1
)
aug_gen = ImageDataGenerator(
```

```
   rotation_range=15,
horizontal_flip=True,
validation_split=0.2
)
model_C = build_model(False) history_C = model_C.fit(     aug_gen.flow(X_train, y_train,
batch_size=BATCH, subset='training'),     epochs=EPOCHS,
validation_data=aug_gen.flow(X_train, y_train, batch_size=BATCH, subset='validation'),
verbose=1
)
model_D = build_model(True) history_D = model_D.fit(
aug_gen.flow(X_train, y_train, batch_size=BATCH, subset='training'),
epochs=EPOCHS,     validation_data=aug_gen.flow(X_train, y_train,
batch_size=BATCH, subset='validation'),     verbose=1
)
plt.figure() plt.plot(history_A.history['val_accuracy'])
plt.plot(history_B.history['val_accuracy'])
plt.plot(history_C.history['val_accuracy'])
plt.plot(history_D.history['val_accuracy']) plt.legend(["No Reg",
"Dropout", "Augmentation", "Dropout+Aug"]) plt.xlabel("Epoch")
plt.ylabel("Validation Accuracy") plt.title("Effect on Overfitting")
plt.savefig("overfitting_accuracy.png", dpi=300) plt.show()
```

14. import tensorflow as tf from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Flatten, Dense, Dropout from tensorflow.keras.datasets
import mnist from tensorflow.keras.utils import
to_categorical import matplotlib.pyplot as plt import os

```
(X_train, y_train), (X_test, y_test) = mnist.load_data() X_train,
X_test = X_train / 255.0, X_test / 255.0 y_train_onehot =
to_categorical(y_train, num_classes=10) y_test_onehot =
to_categorical(y_test, num_classes=10) activations = ['relu',
'tanh', 'sigmoid']

loss_functions = ['sparse_categorical_crossentropy', 'mean_squared_error']
results = {} os.makedirs("plots", exist_ok=True) def
build_model(activation='relu', loss_fn='sparse_categorical_crossentropy'):
   model = Sequential([
      Flatten(input_shape=(28,28)),
      Dense(128, activation=activation),
      Dropout(0.3),
      Dense(64, activation=activation),
      Dense(10, activation='softmax')
   ])
   model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy'])
return model for act in activations:
   print(f"\nTraining with activation: {act}")
model = build_model(activation=act)
history = model.fit(       X_train, y_train,
epochs=5,       batch_size=128,
validation_split=0.2,       verbose=2
   )
   loss, acc = model.evaluate(X_test, y_test, verbose=0)
results[f"{act}_activation"] = {'history': history.history, 'test_acc': acc}
plt.figure()    plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.title(f'Activation: {act}')
```

```
   plt.xlabel('Epoch')     plt.ylabel('Accuracy')
plt.legend()     plt.grid(True)
plt.savefig(f"plots/activation_{act}.png", dpi=300)
   plt.close() for loss_fn in
loss_functions:
   print(f"\nTraining with loss function: {loss_fn}")
model = build_model(loss_fn=loss_fn)     if
loss_fn == 'mean_squared_error':
y_train_input = y_train_onehot        y_test_input
= y_test_onehot
   else:
      y_train_input = y_train
y_test_input = y_test
history = model.fit(
X_train, y_train_input,
epochs=5,
batch_size=128,
validation_split=0.2,
verbose=2
   )
   loss, acc = model.evaluate(X_test, y_test_input, verbose=0)
results[f"{loss_fn}_loss"] = {'history': history.history, 'test_acc': acc}
   plt.figure()     plt.plot(history.history['accuracy'],
label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.title(f'Loss Function: {loss_fn}')
plt.xlabel('Epoch')
```

```
   plt.ylabel('Accuracy')     plt.legend()
plt.grid(True)
plt.savefig(f"plots/loss_{loss_fn}.png", dpi=300)
plt.close()
print("\n=== Summary of Test Accuracies ===") for
key in results:
   print(f"{key}: {results[key]['test_acc']*100:.2f}%")
```

15. import tensorflow as tf from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Flatten, Dense, Dropout from tensorflow.keras.datasets
import mnist
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, CSVLogger import os

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0 def
build_model():
   model = Sequential([
      Flatten(input_shape=(28,28)),
      Dense(128, activation='relu'),
      Dropout(0.3),
      Dense(64, activation='relu'),
      Dense(10, activation='softmax')
   ])
   model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
return model
model = build_model() os.makedirs("callbacks_models",
exist_ok=True)
```

```
callbacks = [
   EarlyStopping(monitor='val_loss', patience=3, verbose=1, restore_best_weights=True),
   ModelCheckpoint(filepath='callbacks_models/best_model.h5', monitor='val_accuracy',
save_best_only=True, verbose=1),
   ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbose=1),
   CSVLogger('callbacks_models/training_log.csv', append=False)
]
history = model.fit(
X_train, y_train,
validation_split=0.2,
epochs=20,
batch_size=128,
callbacks=callbacks,
verbose=2
)
loss, acc = model.evaluate(X_test, y_test) print(f"Test
Accuracy: {acc*100:.2f}%")
```

16.

```
import tensorflow as tf from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import
Flatten, Dense, Dropout from tensorflow.keras.datasets
import mnist import matplotlib.pyplot as plt import os
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0 def
build_model():
   model = Sequential([
```

```
        Flatten(input_shape=(28,28)),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
return model model = build_model() os.makedirs("plots", exist_ok=True) history = model.fit(
X_train, y_train,     validation_split=0.2,     epochs=15,     batch_size=128,     verbose=2
)
plt.figure() plt.plot(history.history['accuracy'], label='Training
Accuracy') plt.plot(history.history['val_accuracy'], label='Validation
Accuracy') plt.title('Training vs Validation Accuracy')
plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.legend() plt.grid(True)

plt.savefig('plots/accuracy_curve.png', dpi=300)
plt.show() plt.figure() plt.plot(history.history['loss'],
label='Training Loss') plt.plot(history.history['val_loss'],
label='Validation Loss') plt.title('Training vs Validation
Loss') plt.xlabel('Epoch') plt.ylabel('Loss') plt.legend()
plt.grid(True) plt.savefig('plots/loss_curve.png', dpi=300)
plt.show() loss, acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {acc*100:.2f}%")
```

1. Manually draw a Fully Connected Feed-forward Neural Network (FCFNN) having 8
neurons in the input layer, 10 neurons in the output layer, and three hidden layers
having 4, 8, 4 neurons, respectively.
2. Write a report in pdf format using any Latex system after:
● drawing a Fully Connected Feed-forward Neural Network (FCFNN) according to
your preferences.
● implementing it using Tensforflow.Keras.
3. Write a report in pdf format using any Latex system after:
● building FCFNNs for solving the following equations:
i. y = 5x + 10
ii. y = 3x
2 + 5x + 10
iii. y = 4x
3 + 3x
2 + 5x + 10
● preparing a training set, a validation set and a test set for the above equations.
● training and testing FCFNNs using your prepared data.
● plotting original y and 'predicted y'.
● explaining the effect of "power of an independent variable" on the architecture of
your FCFNN and the amount of training data.
4. Write a report in pdf format using any Latex system after:
● building an FCFNN based classifier according to your preferences about the
number of hidden layers and neurons in the hidden layers.
● training and testing your FCFNN based classifier using the:

○ Fashion MNIST dataset.
○ MNIST English dataset.
○ CIFAR-10 dataset.
5. Write a report in pdf format using any Latex system after:
● building a Convolutional Neural Network (CNN) based 10 class classifier
● training and testing the classifier by using the the:
○ Fashion MNIST dataset.
○ MNIST English dataset.
○ CIFAR-10 dataset.
6. Write a report in pdf format using any Latex system after:
● Prepare an English handwritten digit dataset by collecting hand written data and
splitting into the training set and test.
● Retrain FCFNN using your training set with the training set of the MNIST English
digit dataset.
● Evaluate your FCFNN using your test set along with the test set of the MNIST
English dataset.
7. Write a report in pdf format using any Latex system after:
● training and testing a CNN based classifier using images captured by you and
your group mates using mobile phones.
● mentioning total training time, testing time per sample, amount-of-data vs
performance, epoch vs performance, model size (i.e., number of parameters) vs
performance and some other observations that you think are interesting and

informative.
8. Build a CNN based classifier having architecture similar to the classical VGG16.
9. Write a report on how feature maps of different convolutional layers look when you pass
your favourite image through your three favourite pre-trained CNN classifiers..
10. Write a report in pdf format using any Latex system after:
● training a binary classifier, based on the pre-trained VGG16, by transfer learning
and fine tuning.
● showing the effect of fine-tuning:
i. whole pre-trained VGG16
ii. partial pre-trained VGG16
11. Discuss the feature extraction power of your favorite CNN pretrained by the ImageNet
dataset before and after transfer learning by the MNIST digit dataset after plotting high
dimensional feature vectors on 2D plane using the following two dimension reduction
techniques:
● Principal Component Analysis (PCA)
● t-distributed Stochastic Neighbor Embedding (t-SNE)
12. Write a report by discussing the effect of different data augmentation techniques on your
CNN based classifiers.
13. Show the effect of dropout layer, data augmentation techniques on overfitting issues of
your CNN based classifier.

14. Write a report by discussing the effect of the following issues on the classifier's performance:

- **different activation functions in hidden layers**
- **different loss functions**

15. Write a report by describing how different callback functions can make your training process better.

16. Write a report describing how monitoring performance curves for both the training set and the validation set based on the target metric (e.g., 'accuracy') and 'loss' metric can improve your hyperparameter training.