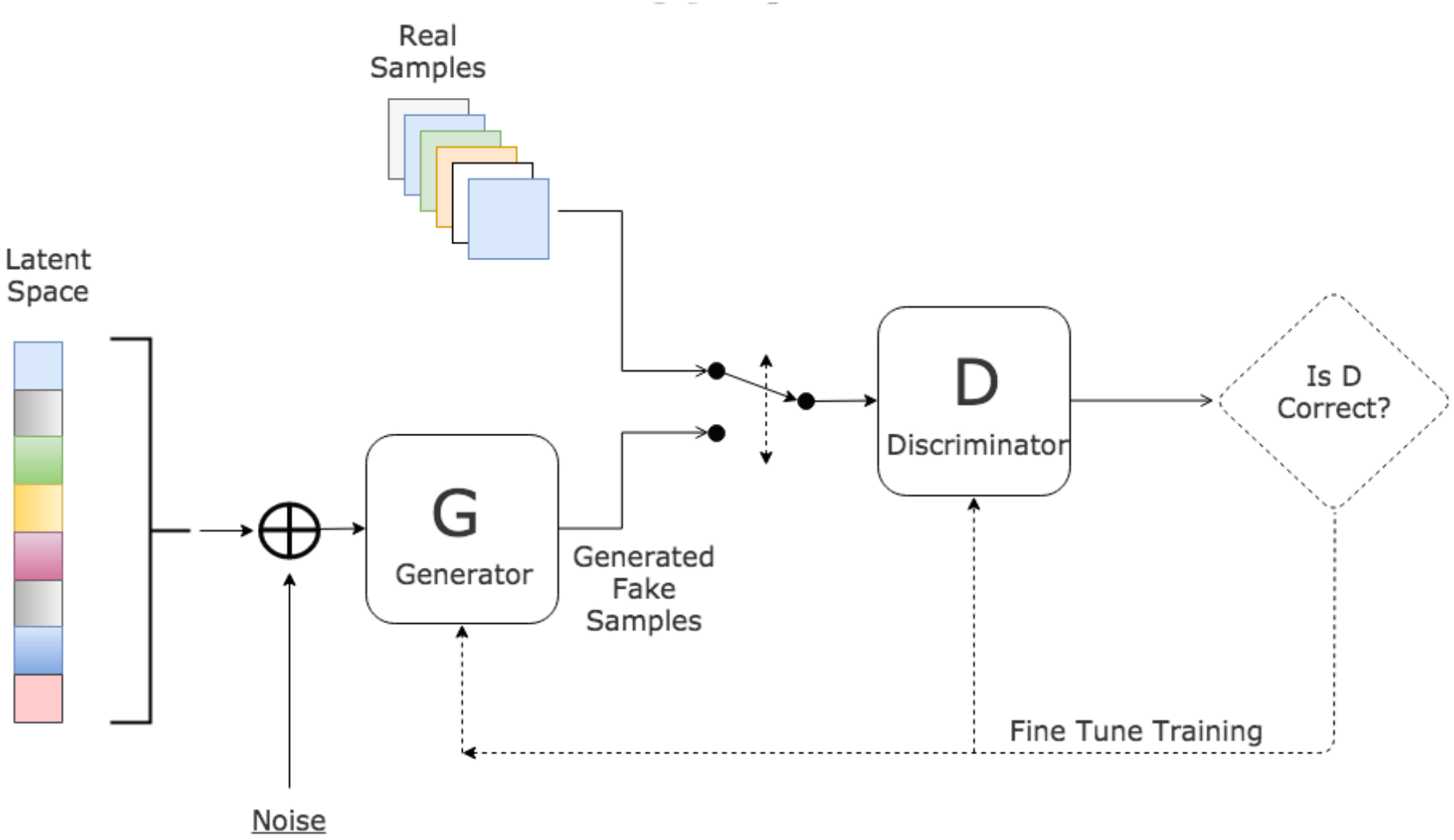
GAN in KERAS

GANs (Generative Adversarial Networks):

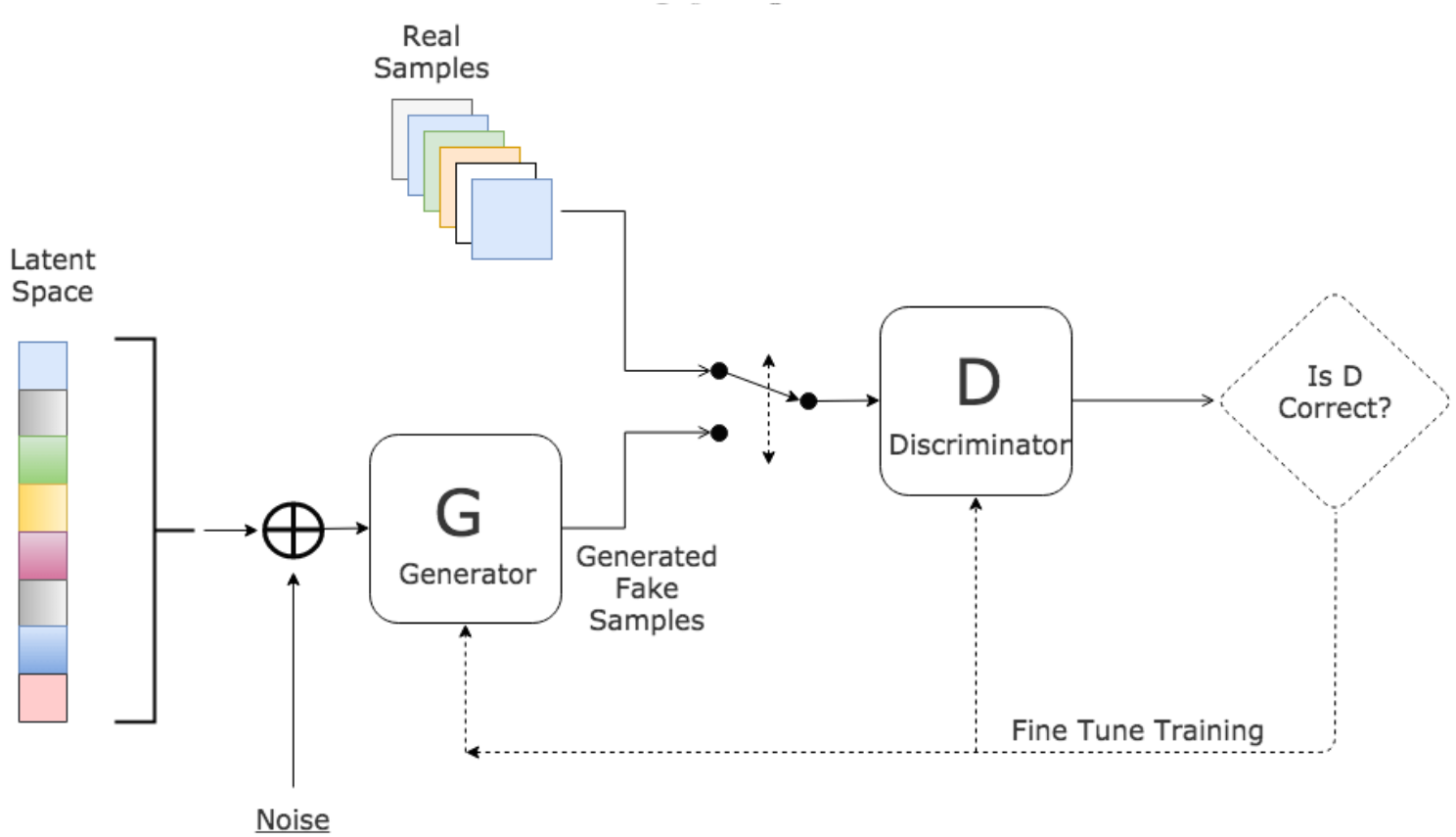


Training Pipeline in GANs:

- The generator takes in random numbers and returns an image.
- This generated image is fed into the discriminator alongside a stream of images taken from the actual dataset.
- The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

Training Pipeline in GANs:

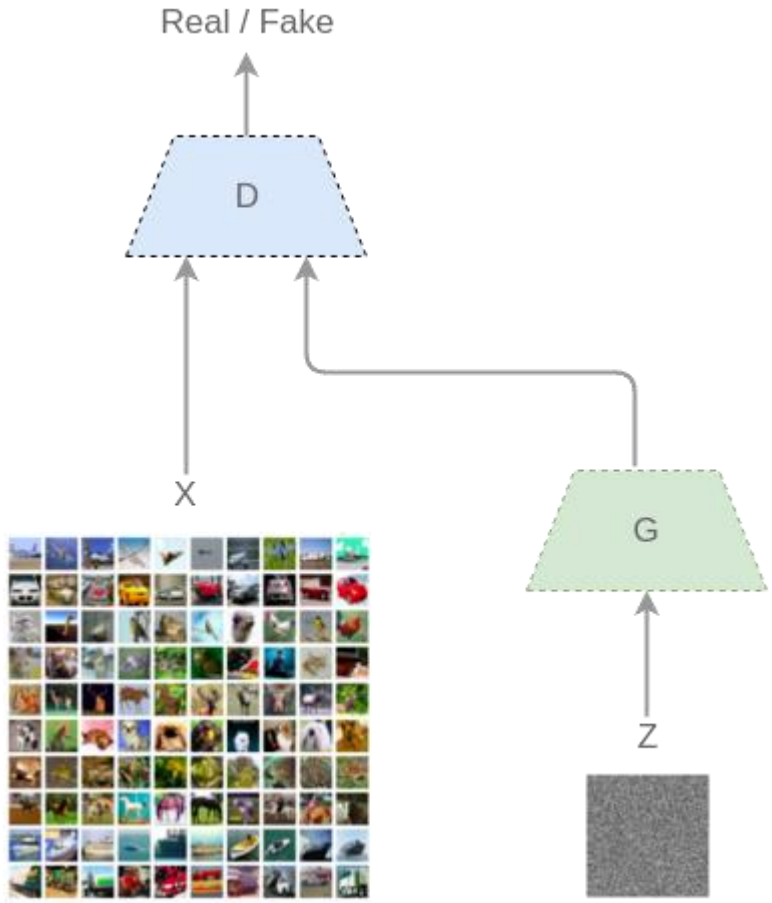
You can think of a GAN as the combination of a counterfeiter and a cop in a game of cat and mouse. the counterfeiter is learning to pass false notes, and the cop is learning to detect them.



Hands-on Discriminator:

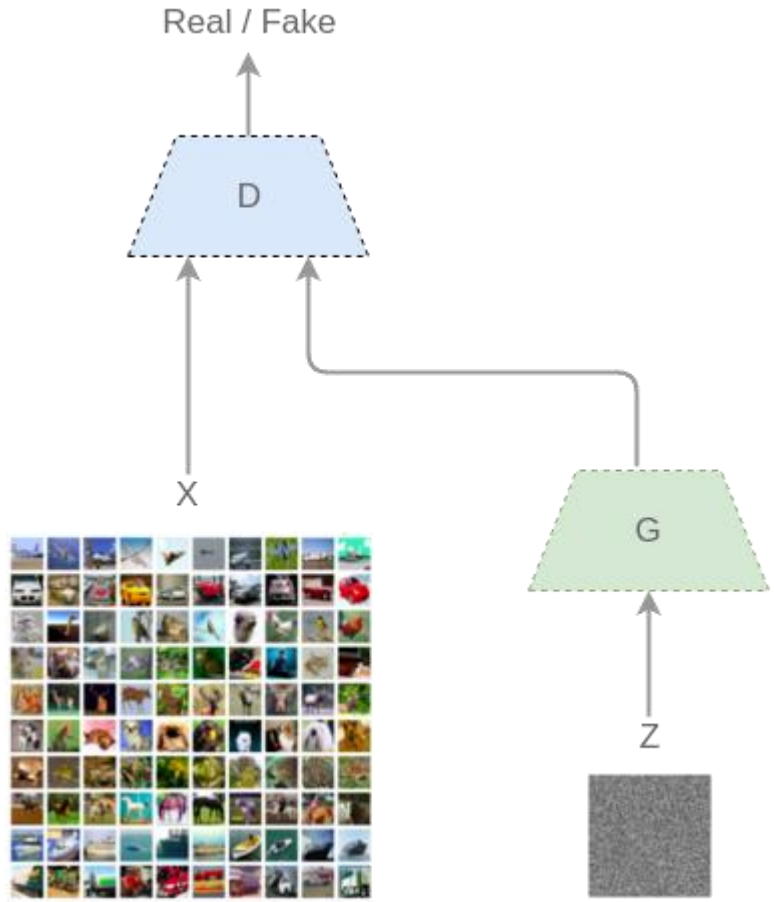
```
model = Sequential()
model.add(Flatten(input_shape=self.img_shape))
model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2))
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.2))
model.add(Dense(1, activation='sigmoid'))
model.summary()

img = Input(shape=self.img_shape)
validity = model(img)
```



Hands-on Discriminator:

```
model = Sequential()
model.add(Dense(256, input_dim=self.latent_dim))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(1024))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(np.prod(self.img_shape), activation='tanh'))
odel.add(Reshape(self.img_shape))
noise = Input(shape=(self.latent_dim,))
mg = model(noise)
```



Compile function important rule in GAN:

```
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy',    optimizer=optimizer,   metrics=['accuracy'])
```

```
z = Input(shape=(latent_dim,))
img = generator(z)

discriminator.trainable = False
validity = discriminator(img)

combined = Model(z, validity)
combined.compile(loss='binary_crossentropy', optimizer=optimizer)
```

Different Training Strategy:

```
for epoch in range(epochs):
        idx = np.random.randint(0, X_train.shape[0], batch_size)
        imgs = X_train[idx]
        noise = np.random.normal(0, 1, (batch_size, latent_dim))
        gen_imgs = generator.predict(noise)

        d_loss_real = discriminator.train_on_batch(imgs, valid)
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        noise = np.random.normal(0, 1, (batch_size, latent_dim))
        g_loss = combined.train_on_batch(noise, valid)

        print ("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100*d_loss[1], g_loss))
```